DEVELOPMENT OF AN INCOMPRESSIBLE NAVIER-STOKES SOLVER FOR MOVING BODY PROBLEMS USING AN OVERSET MESHING APPROACH

JESSICA MACKENZIE

PhD 2017

Centre for Mathematical Modelling and Flow Analysis
School of Computing, Mathematics and Digital Technology

DEVELOPMENT OF AN INCOMPRESSIBLE NAVIER-STOKES SOLVER FOR MOVING BODY PROBLEMS USING AN OVERSET MESHING APPROACH

JESSICA MACKENZIE

A thesis submitted in partial fullfilment of the requirements
of the Manchester Metropolitan University
for the degree of Doctor of Philosophy

2017

Contents

List of Tables

List of Figures

Abstract

An overset meshing approach is an effective method of simulating fluid flow involving multiple moving bodies. It consists of minor meshes representing solid objects, which overlap a Cartesian background grid, allowing bodies to move arbitrarily whilst retaining communication between grids. However, a hole beneath each overlapping mesh must be cut from the background grid, leaving a small overlap. Current hole-cutting methods tend to be complex with some requiring extensive user knowledge and input. Since the hole must be re-cut regularly for moving body problems, it can become very time-consuming.

An original approach for performing a hole-cut has been implemented by employing the Cartesian cut-cell method. This method would ordinarily be used to cut the boundary of a solid object from a single Cartesian grid, as an alternative to the overset grids approach. Thus, the treatment of cut-cells has been modified for its new purpose of hole-cutting. The cut-cell method is already a well-established technique for cutting a Cartesian grid, and is fully automated. It has not been used for hole-cutting previously within the literature and offers a very different hole-cut to existing techniques; It cuts though cells rather than around them, simplifying the cutting process and providing a smooth cut.

This approach has been applied to an incompressible Navier-Stokes solver for viscous single fluid flow. Unstructured, triangular minor meshes are used due to

their ability to represent complex geometries accurately. An explicit time integration method is used on these minor meshes, but an implicit integration method is implemented on the Cartesian background mesh. This new hybrid of integration methods was found to significantly reduce the CPU time in comparison to using a fully explicit method. The solver has been validated using benchmark tests, including a lid driven cavity and flow past a stationary/oscillating cylinder. The results obtained were found to be in good quantitative agreement with published numerical results.

The solver was developed for 2-phase flow problems. However, during the initial validation test, convergence issues were encountered, which meant a sufficient solution could not be obtained.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Acknowledgements

I would like to thank, Ling for all of the help, guidance and time you have given to support me, Clive for your detailed feedback on my written work, Pedro for your technical help with software packages, and everybody at the CMMFA for allowing me to present my research at the meetings and providing feedback and advice.

Finally, I would like to thank Natalie for proof reading my thesis and providing unlimited moral support.

Chapter 1

Introduction

1.1 Motivation

Wave energy converters (WECs) and deep offshore floating wind turbines offer the potential for far greater output of energy than tidal stream turbines or near shore mounted turbines. Since these devices are located in deep offshore waters, they often operate in a harsh marine environment with moving or floating parts interacting with strong waves [3]. This means the hydrodynamic performance, stability and overall survivability of these devices is of high importance and must be evaluated in the design process.

As well as experimental studies, theoretical and numerical analysis has become a fundamental part of both the design and optimisation processes of these types of device. In particular, computational fluid dynamics (CFD) has become increasingly popular. CFD uses numerical methods to solve fluid flow problems.

Computationally modelling WECs and deep water offshore turbines requires the use of a two-fluid (air and water) solver, which is capable of handling complex geometries and possible arbitrary motion of floating parts and breaking waves.

Members of the Centre for Mathematical Modelling and Flow Analysis (CMMFA)

have been developing computational simulations of fluid flow interacting with moving/floating objects [4]. The CMMFA have produced the in-house code, AMAZON-SC, which uses finite volume methods and a Riemann-based solver for solutions of incompressible Navier-Stokes equations.

Finite volume methods are used widely within CFD. These methods have advantages over other types of Eulerian methods, such as finite difference or finite element methods. Firstly, finite volume methods conserve variables such as the flux from one mesh cell to its neighbour [5]. Secondly, these methods can be utilised on unstructured meshes with more ease than finite difference methods. Although the generation of unstructured meshes tends to be computationally expensive, they are extremely useful and advantageous over their structured counterparts. In particular, they can be generated to better suit complex geometries and improve the solution accuracy. Mavriplis [6] gives an extensive review of a range of techniques for generating such grids, currently being used within the CFD community.

AMAZON-SC currently adopts a structured mesh technique for use with the Cartesian cut cell method [3]. This cut cell method allows for simulations of problems involving multiple moving bodies. An alternative method is an overset grid approach [7],[8],[9],[10],[11]. As well as the ability to simulate multiple moving bodies, this method can handle complex geometries while maintaining a high quality mesh, unlike the cut cell method. The development of overset grids is discussed by Chan [12].

It is hoped that a solver which employs both unstructured and overset meshes can be created and combined with AMAZON-SC. The resulting hybrid code can be expected to solve problems involving both complex geometries and wave interactions with moving bodies of arbitrary motion. It should be robust and capable of dealing with the complexities of real-world applications. The solver is

to be developed for use within the CMMFA, to aid their research in moving body problems.

1.2 Aims & Objectives

The overall aim of the project is to develop an incompressible flow solver, which uses a cell centred finite volume approach. It will be based on a combination of structured and unstructured meshes with an overset meshing capability. This will allow for simulations of fluid interactions with moving bodies to be performed efficiently, with complex geometries represented accurately. This project will be broken down into the following objectives, achievable over the three-year period:

1. Implement a single fluid flow solver on an unstructured, 2D mesh using a cell centred finite volume approach.

2. Develop the overset meshing capability of the flow solver through the use of algorithms for automatic detection of overlapping regions between block meshes and data interpolation for boundary conditions.

3. Develop the solver for two-phase (air & water) flow simulations.

1.3 Layout of Thesis

This thesis gives a detailed account of the development of a numerical flow solver, written in FORTRAN 90. It is structured in a way that matches the order of the development of the solver, with results from validation tests provided at each development stage. **Chapter 2** contains a detailed literature review of the the subject, conducted to gain relevant expertise and ensure a gap in the field had been found. This has been an ongoing process throughout, to keep

knowledge up to date and aid the development process of the solver. **Chapter 3** contains numerical methods and validation tests for the unstructured mesh solver first developed. Initially the Shallow Water equations were chosen as a way of beginning the solver development process by employing current knowledge, enabling experience to be gained within the programming language. This was then progressed to an incompressible Navier-Stokes solver as intended. The next development goal was overset capability, which is discussed in **Chapter 4**. Again, both numerical methods and validation tests are provided here. This is a key chapter within the thesis, as it contains specifics of the main novel aspect of the research project, which is the cut-cell hole-cutting method. It also includes details of a new hybrid of time integration methods found to improve efficiency. The final development stage was to allow for 2-phase flow problems to be simulated. The numerical methods and preliminary validation test results for this are provided by **Chapter 5**, with further improvements and and applications discussed in **Chapter 6**.

Chapter 2

Literature Review

2.1 Numerical Methods for CFD

The finite volume method (FVM) is a method for approximating the solution to partial differential equations (PDEs). It was introduced to the field of CFD in 1971 by McDonald [13] and has since become a very popular choice of method. The method works by creating an individual control volume around each conserved variable. The governing equations are then solved directly over this by means of a flux approximation found by integrating over the control volume. As previously stated, this method is favoured over the finite difference method (FDM) and the finite element method (FEM) for problems involving fluids. The FDM is very simple but has more limitations. It cannot be easily applied to unstructured meshes and requires special treatment to enforce conservation [14]. The FEM is a very popular method for problems involving solids but for large fluid flow problems it can become computationally expensive.

The Navier-Stokes equations are the governing equations for real fluid flows. This system of PDEs are a mathematical statement of the conservation of momentum, energy and mass. The solution to these equations includes velocity,

pressure and density. A simpler form of the system of equations is the incompressible Navier-Stokes equations, which rules out sound propagation and gives accurate description of low speed water or air flows. The incompressible Navier-Stokes equations can be solved by a number of numerical techniques including the projection method [15],[16] and the pseudo-compressibility method [17],[18]. The advantage of the latter is that the pressure and velocity fields are directly coupled at the same time level to produce a hyperbolic system of equations. This means that standard upwind finite volume schemes for the compressible Navier-Stokes equations can be implemented on the incompressible form of the equations. The disadvantage is that it is not suitable for computing solutions which are time-dependent. This issue has been resolved by the introduction of an artificial time integration method. This process allows a steady-state solution to be found in artificial-time for each iteration in real-time.

2.2 Free Surface Flows

A free surface flow is a flow which has a fluid interface acting under gravity. Simulating free surfaces is a highly important part of CFD. This is because the location of the free surface defines a flow boundary. When dealing with single-phase flow, this boundary is simply the edge of the computational domain, but when there is two-phase flow present it can be a boundary dividing the two fluids, such as air and water.

One method for dealing with free surface flows is the surface-fitting method [19],[20],[21]. This is a simple and efficient method, which treats the free surface as a moving upper-boundary and calculates a solution only for the liquid region below. Since the method has limited ability and cannot simulate two-phase flow separated by a free surface, it is best suited to simple problems which do not

contain overturning waves, whereby air pockets could arise.

An alternative technique is the free surface capturing method [22],[23]. This approach treats the free surface as a contact discontinuity in density and allows its position to automatically be captured within the numerical solution [24]. This is achieved through the enforcement of conservation laws, similar to the shock-capturing method for compressible flow. It eliminates the need for any special procedure, such as surface tracking or surface fitting. The surface capturing method benefits from its simplicity and handles flow features such as entrapment of one fluid into another with ease. However, if the mesh is not sufficiently fine around the free surface inaccuracies can occur through numerical dissipation when dealing with long calculations.

A third approach is to use the level set method [2]. This is similar to the volume of fluid (VOF) method [25],[26], where a transport equation is solved at each time-step and the shape of the free surface is reconstructed. Instead of tracking the surface in this manner, the level set method simply defines its zero level set as the free surface. This means that an exact representation of the free surface should be found at each time-step with reduced numerical dissipation in long calculations.

2.3 Meshing Techniques

A mesh is a discretisation of a physical domain prior to a computer simulation. The mesh quality [27] can have a big impact on the accuracy of the results of a simulation. It is therefore very important to generate a mesh which can give a good representation of the domains geometry. The quality of a mesh can be measured in terms of its skewness, smoothness and aspect ratio. Skewness refers to the difference in size of the angles within a cell. For a triangular mesh, the

desired value for skewness is zero, which would mean it is an equilateral triangle. Very acute angles within a mesh can lead to inaccuracies within the solution. Smoothness is a measure of the change in area from one cell to a neighbouring cell. A gradual change in cell size throughout a mesh is desired, as sudden jumps in size can lead to errors in the solution. The aspect ratio of a cell is the ratio of a cells longest side to its shortest side. The ideal value for this is one, which means the sides are of equal lengths. A large value signifies a stretched cell, which again can cause inaccuracies. Another important factor to consider is the computational expense of the mesh, in terms of the mesh generation itself, as well as any effects the mesh has on the computations for the solution procedure.

2.3.1 Structured Meshes

Structured meshes are the most simple and tend to be the least computationally expensive. This is because they can be represented by a coordinate system with an index given to each dimension. This coordinate system could be of Cartesian or Polar form. Polar coordinates are used for curvilinear grids, which allow for curved boundaries. This includes internal boundaries, where the mesh is generated around a body. However, it is not possible to represent body geometries on Cartesian grids in this body-fitted manner.

Immersed Boundary Methods

Immersed boundary methods (IBM) are an alternative to body fitted structured or unstructured grids. The body geometry is represented by a discrete set of body forces rather than directly by the mesh. It is this separation of grid generation and body representation that makes the method desirable. Most commonly, a

Cartesian mesh is used for simplicity. The forcing terms are added to the governing equations so that local boundary conditions are satisfied at the immersed boundary [28]. Originally immersed boundary methods were introduced by Peskin [29] as a method of simulating blood flow in the human heart. Since then, the methods have been adapted and developed for a wide range of applications, including moving body problems within ocean engineering. Immersed boundary methods can be separated into two main categories; diffused [29],[30] and sharp-interface methods [31],[32]. The main drawback of the methods is their inability to deal with moving bodies undergoing large displacements. In these cases, spurious oscillations [33] in the pressure field are observed.

Cartesian Cut Cell Method

The Cartesian cut cell method is an alternative to immersed boundary methods but is of similar nature. It was originally developed for use in aerospace for simulations involving complex geometries [34],[35],[36],[37]. The method takes a Cartesian grid and cuts directly through cells to form the geometry of a flow feature or solid object. The advantages of this approach have been outlined by Causon et al. [38] as follows:

1. 'There is no mesh generation in the conventional sense. This is replaced by relatively straightforward calculations for the boundary segment intersections with the background Cartesian mesh.'

2. 'The method can be implemented very efficiently such that only data relating to cells defining boundaries and the flow domain need be held in memory. Consequently, a long narrow meandering flow domain like a river estuary can be handled just as efficiently as a region which is nearly rectangular.'

3. 'The majority of the flow domain is overlaid with a regular Cartesian mesh

so that loss of solution accuracy due to any pathological cases involving excessively stretched or skewed cells is avoided.'

4. 'Moving flow boundaries can be accommodated by recomputing cell-boundary intersections as boundaries move, rather than re-meshing the whole flow domain or rather large portions of it; furthermore, the amplitude of boundary motion is unrestricted.'

5. 'When used in conjunction with mesh adaptation adapting to irregular static or moving boundaries and/or static or moving flow features (e.g. bathymetric data or moving fronts), the method can provide fine mesh resolution where required with much larger coarse spacing in regions where spatial gradients are low; thus the method offers the potential to be a highly efficient and a versatile, practical computational modelling tool.'

The object boundary to be cut is defined by a set of poly-lines, $p_i = (x_0, y_0), (x_1, y_1).....(x_n, y_n)$, for the ith region. The knots of these poly-lines are defined in an anti-clockwise direction. The more points that are defined for a region, the smoother the cut will be. However, the distance between the points should always be greater than the cell side length. For a moving body problem these poly-lines are automatically updated using the body velocity. The object is then re-cut from the grid at its new position.

To perform the cut, intersection points need to be found on the grid for each line segment contained within the set of poly-lines defining the object boundary. Before the intersection points can be obtained, the cells in which the start and end points of the line-segment need to be determined. The line segment is defined by a start point (x_s, y_s) and end point (x_e, y_e). The address (I_s, J_s) of the cell containing the start point is computed by equation 2.1, where x_0 and y_0 are the

coordinates of the bottom left corner of the computational domain.

$$I_s = int(\frac{x_s - x_0}{\Delta x}) + 1, \quad J_s = int(\frac{y_s - y_0}{\Delta y}) + 1 \tag{2.1}$$

Next, the quadrant in which the slope, Q, of the line lies in needs to be identified. The four quadrants are $(0, 90°], (90, 180°], (180, 270°], (270, 360°]$. Using this data, the intersection points can be determined[38],[39]. Figure 2.1 illustrates a line-segment on a Cartesian grid, where the points a,b and c are intersection points to be determined.

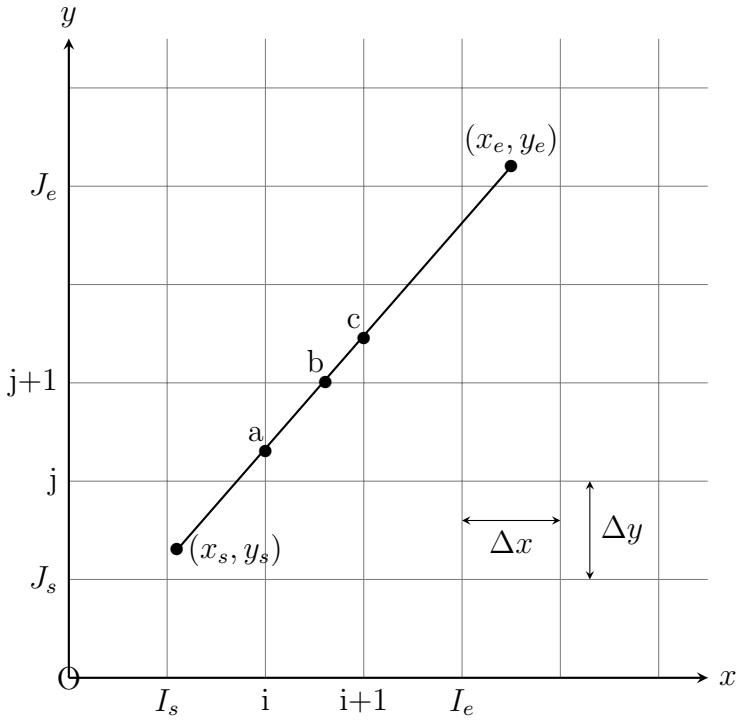


Figure 2.1: Locating intersection points of a line segment.

The point a is where the line-segment enters cell (i,j), but it is also the exit point for the cell (i-1,j), which has already been calculated for this previous cell. Once the point b has been computed, the entry point for the cell (i,j+1) will be known. Hence, it is only the exit points that need to be computed after the

initial intersection point has been determined. For this example, the slope Q is found to be within the quadrant $(0, 90°]$ and a is on the left side of the cell (i,j). This means that the intersection point b will be above and to the right of a and will either intersect the top or right side of the cell. To calculate the intersection point, the following steps provided by Richardson [40] are taken:

1. Calculate
$$\begin{cases} X_b = (I_s - 1)\Delta X \\ Y_b = \frac{y_e - y_s}{x_e - x_s}(X_b - x_s) + y_s \\ Y_a = (J_s - 1)\Delta Y \end{cases} \tag{2.2}$$

2. If $Y_b < Y_a$,then
$$\begin{cases} X_e = X_b \\ Y_e = Y_b \\ I'_s = I_s + 1 \\ J'_s = J_s \end{cases} \tag{2.3}$$

3. If $Y_b >= Y_a$,then
$$\begin{cases} Y_e = Y_a \\ X_e = \frac{x_e - x_s}{y_e - y_s}(Y_e - y_s) + x_s \\ I'_s = I_s \\ J'_s = J_s + 1 \end{cases} \tag{2.4}$$

Once b is known, it is set as the entry point for the cell (i,j+1) and the process is repeated to obtain the exit point c.

This process of finding intersections is repeated for every line-segment defining the object boundary. Upon completion, an enclosed boundary of intersection points is obtained, defining the solid area. Rather than removing sections of the grid, all of the intersection points are stored in an array defining the cut.

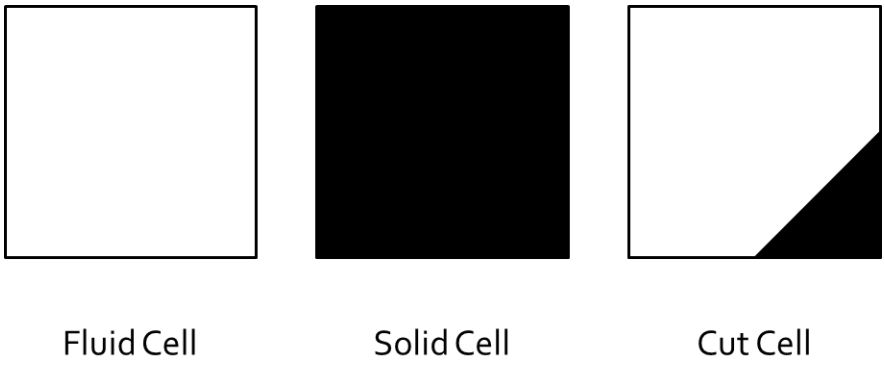


Figure 2.2: Illustration of the four different cell types.

This is then used in the solution procedure. This means the grid never undergoes any deformation or regeneration, even for moving body problems, saving on computational expense and data storage.

Once a set of intersections has been obtained, cells need to be labelled accordingly so that they can be treated appropriately in the solution procedure. There are three different cell labels used, as illustrated in Figure 2.2. Any cells in which intersections are made through are labelled as cut-cells and the whole cells contained within the object are labelled as solid cells. These cells are to be excluded from the solution, which is achieved by setting stored values to zero and implementing solid boundary conditions on the object boundary. Any cells outside of cut regions are labelled as fluid cells. For these cells, the normal solution procedure is used.

For the solution procedure, the cut-cell side length, new cell centroid and cell area must be computed. The cut-cell area is calculated using equation 2.5, as proposed by Clarke et al [41].

$$A_c = \frac{1}{4}(S_x S_y + S_x D_y + S_y D_x - D_x D_y) \tag{2.5}$$

where S_r, S_l, S_t and S_b are the new side lengths of the fluid area of the cut cell.

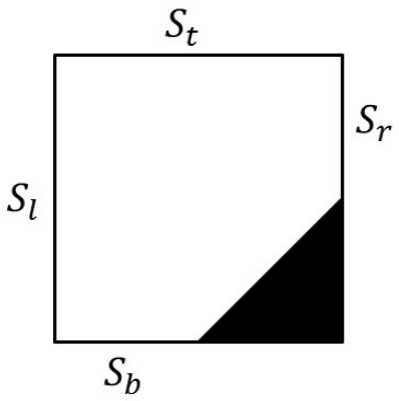


Figure 2.3: Calculation of new cut cell area.

$S_x = S_l + S_r$, $S_y = S_t + S_b$, $D_x = |S_l - S_r|$ and $D_y = |S_t - S_b|$ (see Figure 2.3). To achieve 2nd order spacial accuracy, cell data must be reconstructed at the cell edges using cell gradients. The gradients of a cut-cell (i,j) can be of 2 types: fluid and solid. To calculate the solid gradients, a fictitious or ghost cell, R, is used. This is a receiver cell with values interpolated from the overlapping grid. The fluid gradients U_x^f and U_y^f and solid gradients U_x^s and U_y^s are calculated as follows (see Figure 2.4 ([42])),

$$U_x^f = G\left(\frac{U_{i+1,j} - U_{i,j}}{\Delta x_{i+(1/2),j}}, \frac{U_{i,j} - U_{i-1,j}}{\Delta x_{i-(1/2),j}}\right) \tag{2.6}$$

$$U_y^f = G\left(\frac{U_{i,j+1} - U_{i,j}}{\Delta y_{i,j+(1/2)}}, \frac{U_{i,j} - U_{i,j-1}}{\Delta y_{i,j-(1/2)}}\right) \tag{2.7}$$

$$U_x^s = G\left(\frac{U_R - U_{i,j}}{\Delta x_{i,R}}, \frac{U_{i,j} - U_{i-1,j}}{\Delta x_{i-(1/2),j}}\right) \tag{2.8}$$

$$U_y^s = G\left(\frac{U_{i,j+1} - U_{i,j}}{\Delta y_{i,j}}, \frac{U_{i,j} - U_R}{\Delta y_{i-(1/2),j}}\right) \tag{2.9}$$

where $\Delta x_{i,R} = x_R - x_{i,j}, \Delta y_{j,R} = y_{i,j} - y_R$ and G is the slope limiter function.

To obtain a unique gradient for the cell, a length average of the two types of

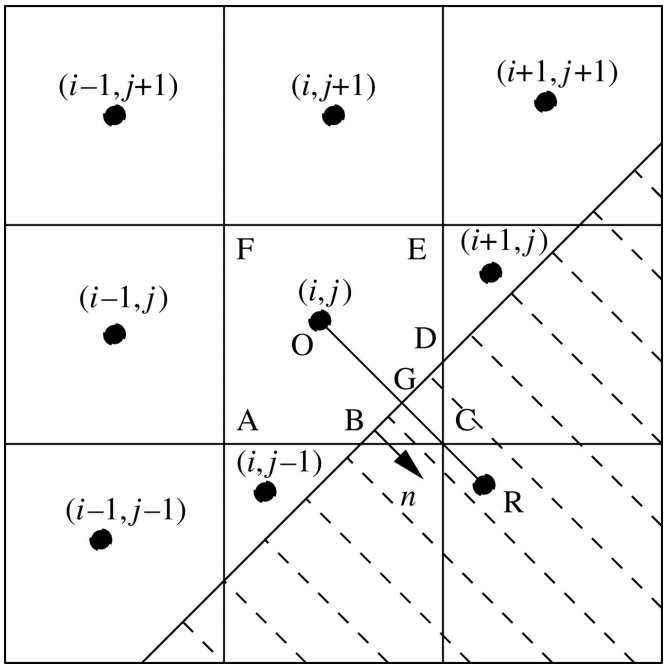


Figure 2.4: Calculation of gradients on cut-cells.

gradient is taken as follows,

$$U_x = \frac{\Delta y_s U_x^s - \Delta y_f U_x^f}{\Delta y} \quad U_y = \frac{\Delta x_s U_y^s - \Delta x_f U_y^f}{\Delta x} \tag{2.10}$$

where $\Delta x_f = |\mathbf{AB}|$, $\Delta x_s = |\mathbf{BC}|$, $\Delta y_f = |\mathbf{CD}|$ and $\Delta y_s = |\mathbf{DE}|$.

A problem found with the cut-cell method was that during the cutting process, small cells were being left as part of the solution domain. These small cells cause instability and accuracy issues. A solution to the stability problem is to reduce the time-step or implement a local time-stepping method to ensure stability is held. But this leaves the accuracy issue caused by the difference in size of the cut cell and it's neighbours. A solution to this was found by Yang et al. [36], who developed a cell merging technique, where any small cells get merged with an appropriate neighbouring cell to form a larger cell. Figure 2.5 provides an example of cell merging. In this case, cell B is considered to be small cell. It is

Figure 2.5: Cell merging.

merged with cell A to form a new cell, C.

There is another problem that remains; Although the method is thought to handle complex geometries well and be advantageous over structured body fitted grids, it still cannot compete with unstructured body fitted grids. This is because the geometry formed by cutting the mesh does not always exactly match the intended geometry and is an approximate due to limitations of the approach. If the geometry contains sharp edges, these will become rounded in the cutting process, causing an error in the approximation.

2.3.2 Unstructured Meshes

Unstructured meshes are more complex than structured. This is because they cannot simply be defined by an index for each direction. Instead, they need to be described by a list of vertices with connectivity information. This means that as well as the computational effort of generating them, there is also the additional expense of storing the required data in an orderly fashion. However, unstructured meshes are a popular choice due to their ability to capture flow features and complex geometries with good accuracy [43]. This is achieved by defining the geometry within the mesh generation process and creating a body fitted grid. Triangular meshes are a popular choice since they can arranged to fit complex geometries, including those with sharp edges.

Many different techniques can be employed for creating an unstructured triangular mesh. Two very popular methods are Advancing Front [44],[45] and Delaunay Triangulation [44],[46].

Advancing Front starts by discretising the boundaries of the geometry as a set of edges, which form an initial front. Edges from this front are then chosen on a priority basis and a triangle is formed from each. The edge is then removed from the front and the two new edges are added to it. This process repeats until there are no edges left in the front. This process is illustrated by Figure 2.6. The main advantage of this method is that it includes an automatic point placement strategy. This optimises the positions of the new points and tends to result in a high-quality mesh. The method also has the advantage of guaranteeing boundaries are held. This is due to the method taking the boundaries as the initial front. The main problem with the technique is its efficiency. The mesh is formed by constructing one triangle at a time. Since in a 2-dimensional mesh there are far more triangle than points, it would be far more efficient to construct the mesh point by point.

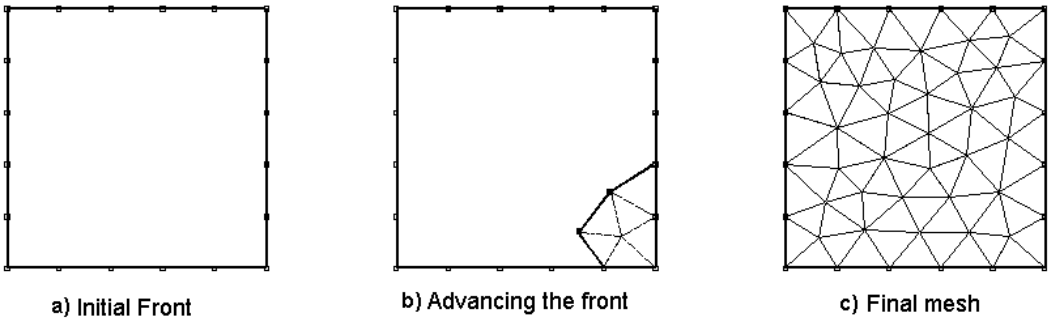
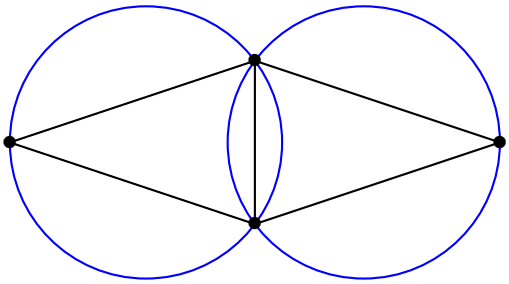


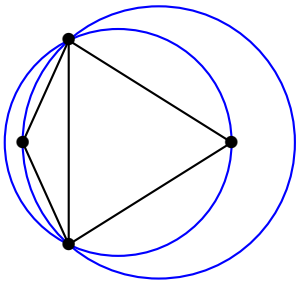
Figure 2.6: Illustration of Advancing Front

Delaunay Triangulation is formed on the basis of the empty circumcircle property. This states that within a triangles circumcircle, there can only ever be vertices which belong to that particular triangle. Figure 2.7a shows an example of

a Delaunay Triangulation, clearly showing that it satisfies the property. Figure 2.7b illustrates a triangulation which does not satisfy the property. The points on the left and right of the diagram are both within circumcircles belonging to another triangle.



(a) Example of Delaunay Triangulation satisfying the empty circumcircle property.



(b) Example of a triangulation which does not satisfy the empty circumcircle property.

The algorithm starts by generating an initial triangulation. This can consist of either a random or a uniform distribution. Mesh points are then added using a point-insertion algorithm. The Bowyer-Watson algorithm [47] is a popular choice for this. For each mesh point inserted, all triangles whose circumcircles contain this point will be deleted. The boundary vertices are then joined to the new point, forming a new triangulation. This method works on a point by point basis unlike advancing front, which gives it the advantage of greater efficiency. Although, Delaunay Triangulation does not guarantee that the boundaries are held. D. Mavriplis [6] gives further explanations of these techniques as well as many others.

As well as the computational expense of unstructured grids, they also suffer in their ability to simulate moving bodies. To simulate this type of problem on an unstructured mesh would either mean continuous mesh regeneration or allowing mesh deformations [48],[49]; both of which are computationally expensive and complex in the solution process.

2.3.3 Overset Meshing

The overset approach is an efficient method of modelling moving objects. It is favourable to the Cartesian cut cell method due to its ability to represent complex geometries whilst maintaining a good quality mesh. The overset mesh consists of sub-grids, which overlap one another; each one representing a different feature of the flow problem or separate body. This means that the features/bodies can be modelled independently of one another and data can simply be exchanged between them.

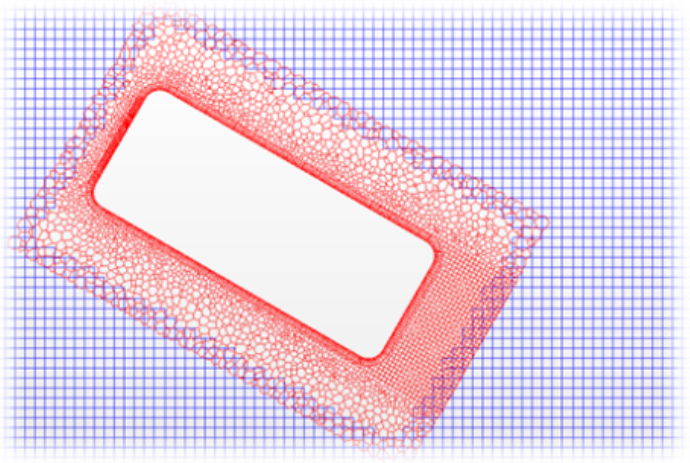


Figure 2.8: An example of overset grids (Blue: background grid, Red: overlapping grid).

Figure 2.8 shows an example of overset grids. In this particular case, there are both structured and unstructured meshes utilised. A Cartesian mesh covers the background of the domain. In front and overlapping this is an unstructured mesh which surrounds a solid object. This object would have the ability to move by means of a moving mesh. Meanwhile, the Cartesian mesh will continue to represent the background without the need for any grid regeneration. This works on the basis that communication is held between the meshes so that data can be interpolated between them. The overset method is perfect for problems

containing multiple moving objects since it can contain as many independent sub-grids as necessary. The use of unstructured grids within the approach results in complex geometries being represented well without the loss of mesh quality or solution accuracy. Panahi and Shafieefar [50] provide a detailed discussion on the overset grid approach, including the implementation of a moving mesh and data interpolation techniques.

Overset grid technology originated from the need to model multi-component systems in the aerospace engineering sector and has been the focus of many research activities over the last 20 years [12]. More recently this technology has also been implemented in some commercial CFD packages such as STAR-CCM+, which has opened up a range of new applications involving arbitrary motion of bodies in fluid. However issues remain in some areas, such as enforcing conservation at grid interfaces and optimisation of parallel processing for a more accurate and efficient implementation of the overset grid methodology.

Hole-Cutting

One of the main issues with the overset approach is the automation of hole-cutting. Hole-cutting is a key stage in the domain connectivity process of overset grids, whereby unwanted grid cells are removed from the solution domain. These cells are removed from a Cartesian background mesh where a solid object lies within an overlapping mesh. No background cells should intersect the area of the solid object but an overlap between the two meshes should be present for interpolation of solution data. When two overlapping meshes intersect, the finer mesh should take priority and the coarser cells should be removed.

In general, a typical hole-cut is performed over three stages as illustrated by Figures 2.9-2.11 [51]. Stage one is to detect any cells that intersect the object boundary. Since these cells all lie partly within the object regions and are labelled

as hole cells and removed from the domain. Stage two is to remove all of the cells that are fully within the object region. This can be performed by scanning through the neighbours of the intersecting cells, and determining whether each one lies inside or outside of the object region. Any neighbouring cells found to be inside the region are labelled as hole cells and these are added to list of cells whose neighbours need to be scanned. Any neighbouring cells found to be outside of the region are labelled as fringe cells, which will be used for interpolation between grids. Stage three is to minimise the overlap between grids by expanding the hole-cut outwards, based on some given overlap criteria.

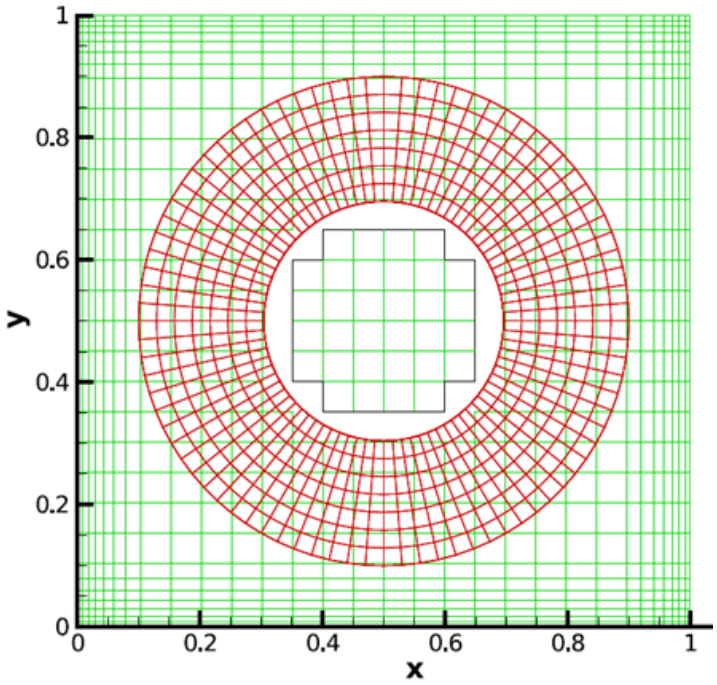


Figure 2.9: Performing a general hole cut (Stage 1: Find intersection cells and remove)

There are many different methods of executing this hole-cutting process, all ranging in complexity, efficiency and levels of user input required. Most hole-cutting techniques can be categorised into one of two types; explicit or implicit cuts.

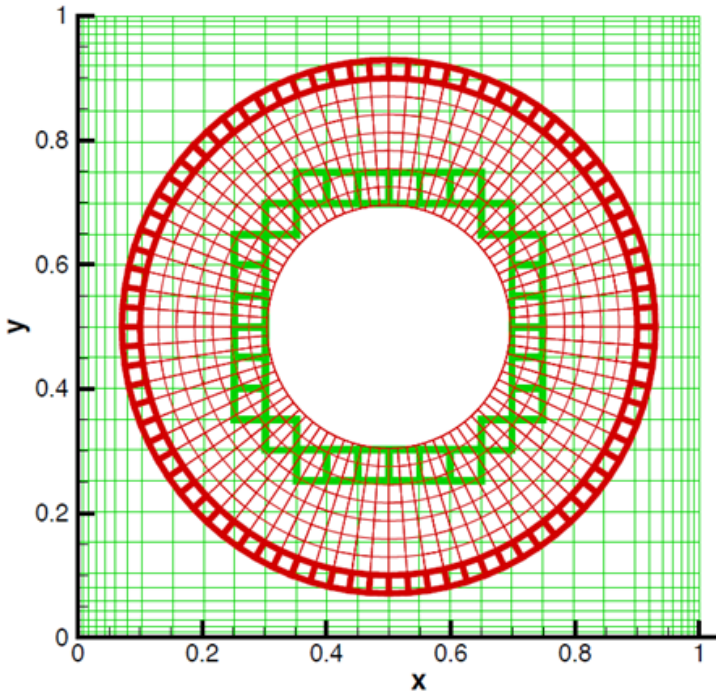


Figure 2.10: Performing a general hole cut (Stage 2: Remove remaining cells within object region).

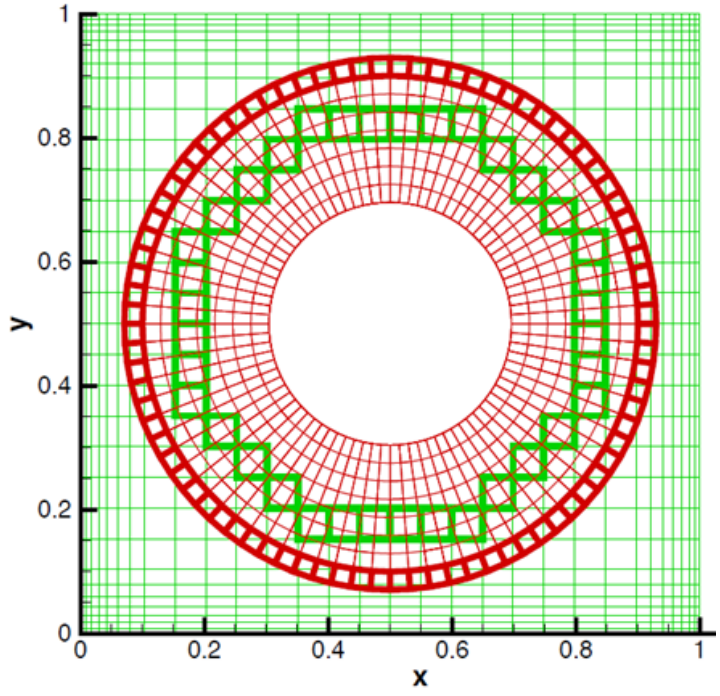


Figure 2.11: Performing a general hole cut (Stage 3: Minimise overlap).

For explicit hole-cutting methods, the way in which the holes are cut must be user specified. This process can be tedious but also relies on the user having expertise. For moving body problems, this would mean user input every time the overlapping grid position is updated. A well used explicit hole-cutting method is the x-rays approach, which was introduced by Meakin [52]; however, this is strictly for three-dimensional grids. It takes a two-dimensional Cartesian map in the x-y plane and lays it over a three-dimensional body. It casts rays in the z direction and stores intersections with the surface of the object. This means that any given three-dimensional point can be found within the two-dimensional map. Comparisons with stored intersection in the z-direction can determine whether the point is inside or outside of the object. The approach proved to be much faster than previous methods and therefore had a big advantage over these for problems involving moving bodies, where the process must be repeated for every time-step. However, the setup process for this method required significant manual inputs and expertise. Inputs include the x-rays themselves and a list of grids to be cut by each x-ray. Advances towards automating this approach were made by Kim et al. [53], whereby two previously manual steps were automated. These were the closure of any open boundaries in the hole-cutter surface and the determination of grid points to be removed; both of which are crucial steps prior to hole-cutting. In 2012, Chan et al.[54] made further improvements to the x-rays approach. A wall-distance function was introduced to offset the hole boundary from the minimum hole, resulting in a better quality mesh. These improvements from the original x-rays method significantly reduced the user requirements. However, the wall-distance function is computationally expensive and therefore slowed the performance of the approach. Further improvements are still required for this method since the wall-distance function has not yet been adapted for relative motion problems in which the x-rays approach becomes advantageous.

Alternatively, implicit hole-cutting methods, which were introduced by Lee & Baeder [55],[56], require little to no user input to perform the hole-cut. However, implicit methods can be computationally expensive. For this reason, they are not suitable for moving body problems, where the hole-cutting process needs to be performed at each time-step. Implicit hole-cutting takes a different approach to typical hole-cuts, in that it does not need to blank out cells that lie within a solid object. Instead, a thick hole fringe layer is used to enclose the object and prevent contamination from those unblanked cells in the computations. Fringe cells are determined through a donor stencil search, which requires searching through all cells within the overlapping region to find the best quality donor cells. This means that the chosen fringe cells can be considered as optimal, with no need for a further stage to optimise or minimise the overlap. However, since the donor stencil search requires all overlapping cells to be considered, this process can be computationally expensive, especially for large overlapping grids.

2.4 Summary

Within this review, it has been found that although structured grids tend to be the most simple and least computationally expensive, they do suffer in their ability to handle complex geometries. Alternatively, unstructured grids handle complex geometries well, but suffer in their ability to handle moving bodies, as well as being computationally expensive and complex in nature. A combination of structured and unstructured meshes using an overset approach would allow for moving body problems, whilst allowing for complex geometries to be represented well through smaller overlapping unstructured meshes. The large portion of the domain being covered by the structured mesh along with the relatively small size of the overlapping meshes should minimise the computational expense. A problem

to overcome, is the automation of hole-cutting for moving body problems; There still remains a need for a fully automatic hole-cutting approach, which is also simple, computationally inexpensive and robust.

Chapter 3

Unstructured Mesh Solver

3.1 Unstructured Mesh Numerical Methods

3.1.1 Introduction

This section contains details of the numerical solver developed for use with unstructured triangular meshes. The solver was written in Fortan 90, using existing numerical methods. It was written for compatibility with the in-house code, with the aim of combining the two codes to develop an overset grids technique.

Firstly, unstructured mesh generation is discussed. This is followed by details of the governing equations (the Shallow Water equations and the Incompressible Navier-Stokes equations) and the numerical methods used in conjunction with both. This includes both spatial discretisation and time integration. Finally, there is a short section outlining how the unstructured data is stored.

3.1.2 Unstructured Mesh Generation

Distmesh [57] is open source MATLAB code. This code has the capability to generate unstructured meshes of high quality. It creates meshes consisting of

triangular cells, using the Delaunay Triangulation algorithm. The geometry of the mesh is defined using a signed distance function. This signed distance function is user-specified, allowing the user to have considerable control over the outcome. It can handle both simple and complex geometries, and requires minimal effort by the user. The program outputs a visual of the changing mesh as it is being refined. Once this is complete, coordinates for all vertices are saved along with a list of their connectivities.

The distmesh code generates meshes for a specified geometry using Delaunay triangulation. The initial triangulation has a uniform distribution, creating a mesh of equilateral triangles. A force-displacement function [58] is used to refine the mesh. The function is solved for equilibrium upon each iteration, resulting in movement of the nodes. Delaunay triangulation then decides on the connectivity of these nodes to form triangles. There are also external forces on the boundaries. For each boundary node, there is a reaction force acting normal to the boundary. This force prevents the node from moving outside of the domain.

The geometry of the domain is defined by a signed distance function, which gives the distance from any point to the geometrys boundary (interface). This distance is signed according to whether it lies within the domain or not. The distances corresponding to points within the domain are negative and outside of the domain are positive. Fedkiw [59] gives an in-depth explanation of signed distance functions.

To generate a 2-dimensional mesh, the function distmesh2d is used. The following MATLAB syntax will call and run the function,

[p,t]=distmesh2d(fd,fh,h0,bbox,pfix,varargin)

The outputs p and t are arrays for the vertex coordinates and connectivity lists respectively. p is an N $\times$ 2 array, where N is the number of vertices within

the mesh. For each vertex, the corresponding x and y coordinates are provided respectively. t is an M × 3 array, where M is number of triangles within the mesh. For each triangle, 3 vertex numbers are listed which connect to form that triangle. These vertex numbers correspond to those within p. The inputs, bracketed on the RHS must be user defined before calling the function. fd is the signed distance function. fh is a function for the desired edge length. h0 is a constant denoting the distance between points in the initial distribution. bbox is an array, [xmin,ymin ; xmax,ymax], containing coordinates for a box to which the domain is bound. pfix is an array containing coordinates for any fixed vertex positions that may be required. varargin is an optional argument, which allows additional parameters to be given to fd and fh. The geometry of a mesh is defined by a user-specified signed distance function, fd. The distmesh code provides the following built-in functions to aid the user in doing this,

drectangle(p,xmin,ymin,xmin,ymax)

dcircle(p,xcentre,ycentre,radius)

dpoly(p,[x1,y1;x2,y2;...xN,yN]

ddiff(d1,d2)

dunion(d1,d2)

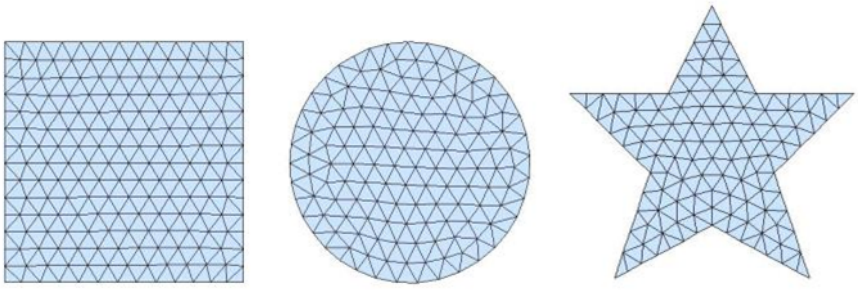


Figure 3.1: Examples of geometries defined by the functions, drectangle, dcircle and dpoly, respectively.

drectangle will create a rectangular geometry. It requires the user to define the size of it by inputting the coordinates for its minimum and maximum points. dcircle will create a circle with a specified centre point and radius. dpoly will create any geometry defined by a set of coordinates. Figure 3.1 shows examples of meshes created using the functions, drectangle, dcircle and dpoly, respectively. The function, ddiff will take the difference between two given geometries. This

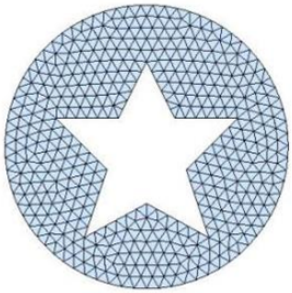


Figure 3.2: Example of the use of the function, ddiff.

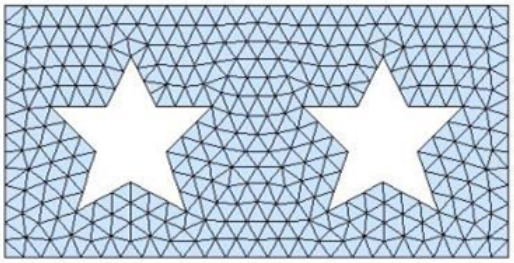


Figure 3.3: Example of the use of the function, dunion.

is very useful for creating meshes whereby solid objects can be represented by deleting a specified geometry from the mesh. The difference has been taken between geometries generated using dcircle and dpoly to give the example shown in Figure 3.2. The dunion function with take the union of two given geometries. Combining this with ddiff allows multiple geometries to be cut out of a mesh. This is illustrated in Figure 3.3.

The edge length of each cell within the mesh is defined by a function fh. The mesh examples given in figs. 3.1 to 3.3 have a uniform distribution. This means the edge lengths are all kept as close as possible to a given constant h0. This is achieved by defining fh as the function huniform, which is built-in to the distmesh code. Sometimes it is advantageous to have a non-uniform distribution to give a finer distribution around the boundaries. This is particularly useful

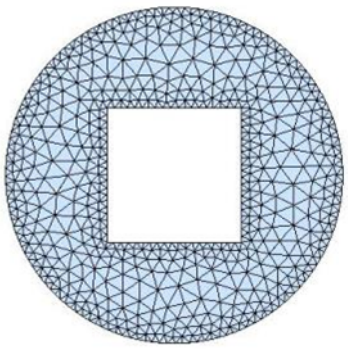


Figure 3.4: Examples of a mesh with a non-uniform distribution.

when complex geometries are involved. It means that the complex geometry can be well represented with smaller cells, whilst leaving the remainder of the region covered with larger cells; Hence, reducing the total number of cells and consequently, the overall run-time. A non-uniform distribution is achievable by incorporating a distance function into the definition of fh. An example of this is illustrated by Figure 3.4.

3.1.3 Governing Equations

The system of partial differential equations (PDEs) in integral form is,

$$\frac{\partial}{\partial t}\iint_{\Omega}\mathbf{U}d\Omega + \oint_{S}\mathbf{F}\cdot\mathbf{n}dS = \iint_{\Omega}\mathbf{H}d\Omega \tag{3.1}$$

where, $\mathbf{U}$ is the vector of conserved variables, Ω is the domain, S is the boundary surrounding Ω, $\mathbf{F}$ is the vector of flux through S, $\mathbf{n}$ is the outward pointing unit normal to S and $\mathbf{H}$ is the vector of forcing functions.

Shallow Water Equations (SWE)

For the Shallow Water Equations [60],[61], **U**, **F** and **H** from equation 3.1 are defined as,

$$\mathbf{F}\cdot\mathbf{n} = (f^I - vf^V)n_x + (g^I - vg^V)n_y,$$

$$\mathbf{U} = \begin{bmatrix} h \\ uh \\ vh \end{bmatrix}, \quad f^I = \begin{bmatrix} uh \\ u^2h + \frac{gh^2}{2} \\ vuh \end{bmatrix}, \quad g^I = \begin{bmatrix} vh \\ uvh \\ v^2h + \frac{gh^2}{2} \end{bmatrix}, \tag{3.2}$$

$$f^V = \begin{bmatrix} 0 \\ hu_x \\ hv_x \end{bmatrix}, \quad g^V = \begin{bmatrix} 0 \\ hu_y \\ hv_y \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 0 \\ gh(S_{0x} - S_{fx}) \\ gh(S_{0y} - S_{fy}) \end{bmatrix}$$

where u and v are the velocity components, n_x and n_y are the components of the normal vector, u_x, u_y and v_x, v_y are the velocity components in the x and y directions respectively. The superscripts I and V denote the inviscid and viscous fluxes respectively, h is the water height or depth, g is the acceleration due to gravity. S_{fx}, S_{fy} and S_{0x}, S_{0y} denote the components of the bottom friction and bed slope respectively.

Incompressible Navier-Stokes Equations (INS)

For the incompressible Navier-Stokes equations U, F and H from equation 3.1 are defined as,

$$\mathbf{F}\cdot\mathbf{n} = (f^I - \frac{1}{Re}f^V)n_x + (g^I - \frac{1}{Re}g^V)n_y,$$

$$\mathbf{U} = \begin{bmatrix} 0 \\ u \\ v \end{bmatrix}, \quad f^I = \begin{bmatrix} u \\ u^2 + p \\ uv \end{bmatrix}, \quad g^I = \begin{bmatrix} v \\ uv \\ v^2 + p \end{bmatrix}, \tag{3.3}$$

$$f^V = \begin{bmatrix} 0 \\ u_x \\ v_x \end{bmatrix}, \quad g^V = \begin{bmatrix} 0 \\ u_y \\ v_y \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 0 \\ -s_x/Fr^2 \\ 0 \end{bmatrix}$$

where p is the dynamic pressure and Re is the Reynolds number, s_x is the free surface slope and Fr is the Froude number. The Froude number is a dimensionless value for the ratio of inertial and gravitational forces.

A Reynolds number is a dimensionless number, used to determine whether flow is laminar or turbulent. It is dependant on the viscosity and calculated as follows,

$$Re = \frac{\rho V L}{\mu} \tag{3.4}$$

where ρ is the density, V is the velocity, L is the characteristic length and μ is the viscosity. A low Reynolds number is the result of high viscosity and describes flow which is laminar. As the viscosity is reduced, the Reynolds number increases and the flow becomes unstable and eventually turbulent. In the present work, only laminar flow will be studied up to a Reynolds number of 1000, for simplicity of the solver.

Inserting equation 3.3 in its current form, into equation 3.1 will not produce a hyperbolic system of equations, where the pressure and velocity are coupled at the same time level. This is because there is no time derivative of pressure present in the continuity equation. To overcome this, a pseudo-compressibility method [18] must be introduced, modifying $\mathbf{U}$, f^I and g^I as follows,

$$\mathbf{U} = \begin{bmatrix} p \\ u \\ v \end{bmatrix}, \quad f^I = \begin{bmatrix} \beta u \\ u^2 + p \\ uv \end{bmatrix}, \quad g^I = \begin{bmatrix} \beta v \\ uv \\ v^2 + p \end{bmatrix} \tag{3.5}$$

where β is the coefficient of pseudo-compressibility.

3.1.4 Spatial Discretisation

The inviscid fluxes, $F^I_{i,j}$ are calculated by adopting Roes flux approximation [62], [63] locally at each cell edge, assuming a 1D Riemann problem in the direction normal to the cell edge, as follows,

$$F^I_{i,j} = \frac{1}{2}[F^I(U^+_{i,j}) + F^I(U^-_{i,j}) - |A|(U^+_{i,j} - U^-_{i,j})]overset \tag{3.6}$$

$$|A| = R|\Lambda|L \tag{3.7}$$

where $U^+_{i,j}$ and $U^-_{i,j}$ are the reconstructed right and left states respectively of the cell edge between cells i and j and A is the flux Jacobian evaluated by Roes average state. Λ is a matrix containing the eigenvalues of A. R and L are the right and left eigenvectors of A, respectively. For the shallow water equations, these matrices, as provided by Anastasiou and Chan [63] are as follows,

$$A = \frac{\partial(F \cdot n)}{\partial Q} = \begin{bmatrix} 0 & n_x & n_y \\ (c^2 - u^2)n_x - uvn_y & 2un_x + vn_y & un_y \\ -uvn_x + (c^2 - v^2)n_y & vn_x & un_x + 2vn_y \end{bmatrix} \tag{3.8}$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \tag{3.9}$$

with $\lambda_1 = un_x + vn_y$, $\lambda_2 = un_x + vn_y - c$ and $\lambda_3 = un_x + vn_y + c$,

$$R = \begin{bmatrix} 0 & 1 & 1 \\ n_y & u - cn_x & u + cn_x \\ -n_x & v - cn_y & v + cn_y \end{bmatrix} \tag{3.10}$$

$$L = \frac{1}{c^2}\begin{bmatrix} -(un_y - vn_x) & n_y & -n_x \\ (un_x + vn_y)/(2c) + \frac{1}{2} & -n_x/2c & -n_y/2c \\ -(un_x + vn_y)/(2c) + \frac{1}{2} & n_x/2c & n_y/2c \end{bmatrix} \tag{3.11}$$

where the average quantities for velocity are, $u = \frac{\sqrt{h_R}u_R + \sqrt{h_L}u_L}{\sqrt{h_R}+\sqrt{h_L}}$ and $v = \frac{\sqrt{h_R}v_R + \sqrt{h_L}v_L}{\sqrt{h_R}+\sqrt{h_L}}$ and for celerity, $c = \sqrt{\frac{g}{2}(h_R + h_L)}$. These are constructed as suggested by Brufau and Garcia [64], For the incompressible Navier-Stokes equations, based on the artificial compressibility approach, the flux Jacobian is,

$$A = \frac{\partial(F \cdot n)}{\partial Q} = \begin{bmatrix} 0 & \beta n_x & \beta n_y \\ n_x & 2un_x + vn_y & un_y \\ n_y & vn_x & un_x + 2vn_y \end{bmatrix} \tag{3.12}$$

The eigenvalues (Λ) and right (R) and left (L) eigenvectors, all provided by Chan and Anastasiou [18] are,

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \tag{3.13}$$

with $\lambda_1 = un_x + vn_y$, $\lambda_2 = un_x + vn_y + c$ and $\lambda_3 = un_x + vn_y - c$,

$$R = \begin{bmatrix} 0 & \beta c & -\beta c \\ n_y & u\lambda_2 + \beta n_x & u\lambda_3 + \beta n_x \\ -n_x & v\lambda_2 + \beta n_y & v\lambda_3 + \beta n_y \end{bmatrix} \tag{3.14}$$

$$L = \frac{1}{c^2}\begin{bmatrix} -(un_y - vn_x) & v\lambda_1 + \beta n_y & -(u\lambda_1 + \beta n_x) \\ -\lambda_3/(2\beta) & n_x/2 & n_y/2 \\ -\lambda_2/(2\beta) & n_x/2 & n_y/2 \end{bmatrix} \tag{3.15}$$

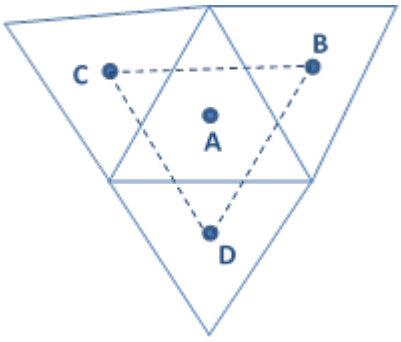


Figure 3.5: A simple integration path for Gauss' theorem.

where c is the pseudo-wave celerity expressed as $[(un_x + vn_y)^2 + \beta(n_x^2 + n_y^2)]^{1/2}$ and the average quantities are $u = (u_R + u_L)/2$ and $v = (v_R + v_L)/2$.

To obtain second order accuracy, the cell variables must first be reconstructed, linearly from the solution data before calculating the left and right Riemann states at the cell edge. The cell values are reconstructed for a given cell with cell centre, A, by the application of equation 3.16, where U_A is the stored cell data, $\mathbf{r}$ is the position vector from the cell centre to any point (x, y) and ∇U_A is the gradient of the solution data of cell A.

$$U(x, y) = U_A + \nabla U_A \cdot \mathbf{r} \tag{3.16}$$

This gradient is estimated using Gauss theorem,

$$\nabla U_A = \frac{1}{A_\Omega} \oint_{\partial A} U \mathbf{n} dS \tag{3.17}$$

where, ∂A is the integration path surrounding A and A_Ω is the area within the integration path. A simple and efficient integration path is given in Figure 3.5. The viscous flux $F_{i,j}^V$, is found by computing the gradient of the velocities in both

the x and y directions at each cell edge as follows,

$$\nabla U_{PQ} = \frac{1}{A_{APBQ}}(A_{BPQ}\nabla U_{APQ} + A_{APQ}\nabla U_{BPQ}) \tag{3.18}$$

where, U_{APQ} and ∇U_{BPQ} are the gradients of the paths APQ and BPQ within Figure 3.6, respectively. These gradients are evaluated by the use of equation 3.18. A_{APBQ}, A_{APQ} and A_{BPQ} are the ares within the paths, $APBQ$, APQ and BPQ respectively.

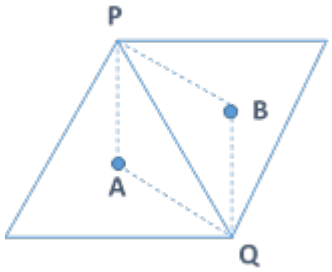


Figure 3.6: The integration paths to compute the gradient of velocities.

The values at the vertices P and Q are found by applying Shepard's method [65]. This is an inverse distance weighted interpolation of the surrounding stored cell centre values, defined by,

$$U = \frac{\sum_{i=1}^{N} w_i U_i}{\sum_{i=1}^{N} w_i}, \quad w_i = \frac{1}{d_i^p} \tag{3.19}$$

where U is the new interpolated values at the given vertex with position X and U_i is the array of stored values at the centre of cell i with position X_i. w_i is the weighting given to a cell centre value calculated using the distance, d_i between X and X_i and N is the total number of neighbouring cell centres to be interpolated from. Figure 3.7 shows a vertex with its neighbouring cell centres and the distances between them, used in determining the weighting of

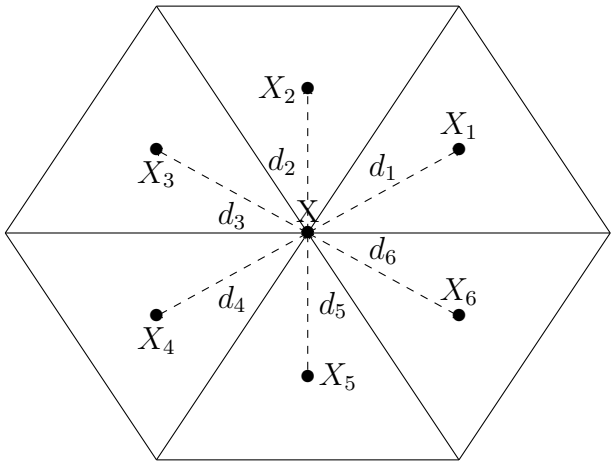


Figure 3.7: Distances used for inverse distance weighted interpolation to obtain vertex values

interpolation. The exponent, p is given a real positive value. For $p < 2$, values further away tend to have too much weighting and dominate the interpolated values. Larger values for p result in the closer values influencing the interpolated value. In finding values at vertices, it is only direct neighbouring cell centre values that are considered. This means the range in distances is relatively small and as a result, the impact p has on weighting is minimised. For this reason, p has been set to 2 to reduce computational effort since $1/d_i^2 = 1/[(x - x_i)^2 + (y - y_i)^2]$.

The problem with the second order scheme is that oscillations can occur. This is due to far more cell values being taken into consideration; the neighbours of each neighbour to a given cell are used in calculating the two Riemann states at each cell edge. The range of values now considered can cause rapid changes in the cell values which ultimately can produce oscillations in the results. To prevent this, a limiter, denoted Φ is used within the reconstruction of cell variables (equation 3.16) to form equation 3.20.

$$U(x, y) = U_A + \Phi \nabla U_A \cdot \mathbf{r} \tag{3.20}$$

$$\Phi = min(\Phi_j), \quad j = k(i) \tag{3.21}$$

$$\Phi_j(r_j) = max[min(r_j, Co), min(r_j Co, 1)] \tag{3.22}$$

$$r_j(U_j) = \begin{cases} (U_0^{max} - U_0)/(U_j - U_0) & if \quad U_j - U_0 > 0, \\ (U_0^{min} - U_0)/(U_j - U_0) & if \quad U_j - U_0 < 0, \\ 1 & if \quad U_j - U_0 = 0 \end{cases} \tag{3.23}$$

$$U_0^{min} = min(U_0, U_{neighbour}), \quad U_0^{max} = max(U_0, U_{neighbour}) \tag{3.24}$$

As stated by equation 3.21, the limiter for a cell is taken to be the minimum limiter in normal direction to each cell side. The limiter for each cell side is found using equation 3.22. r_j is computed by the Barth and Jespersen scheme [66] given by equation 3.23, where U_j is the cell edge value and U_0 is the cell centroid value. When $Co = 1$ this reduces to the minmod limiter and with $Co = 2$ it is the superbee limiter [67].

Boundary Conditions

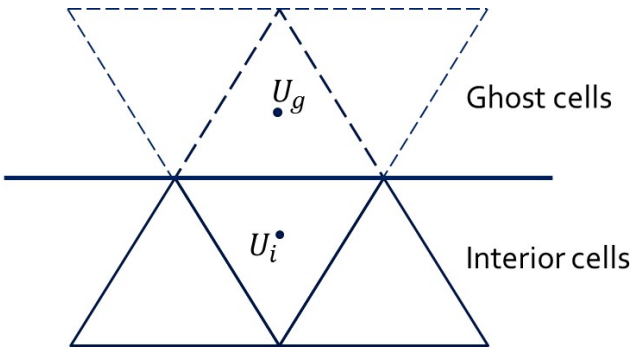


Figure 3.8: Ghost cell method for boundary conditions

To implement boundary conditions, a ghost cell method is employed. This is illustrated by Figure 3.8. Cells are mirrored across the boundary to form ghost cells. For second order accuracy, two sets of ghost cells are required; the cells

with a side lying on the boundary and the neighbours to these cells. Values are given to these ghost cells based on the values of the corresponding interior cells as follows:

For solid wall boundary conditions (no slip),

$$\begin{aligned} p_g &= p_i \\ \mathbf{v}_g &= \mathbf{v}_i - 2(\mathbf{v}_i \cdot \mathbf{n})\mathbf{n} \end{aligned} \tag{3.25}$$

where $\mathbf{v} = [u, v]$ and $\mathbf{n}$ is the outward pointing normal vector.

For transmissive boundary conditions,

$$\begin{aligned} p_g &= p_i \\ u_g &= u_i \\ v_g &= v_i \end{aligned} \tag{3.26}$$

Other boundary conditions are implemented similarly.

3.1.5 Time Integration

For the SWE solver, Eulers implicit forward difference procedure is used to integrate through time,

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \frac{\Delta t}{A_i}(\sum_{j=1}^{NE}(\mathbf{F} \cdot \mathbf{n})_j)_i^n \tag{3.27}$$

where the subscripts i and j denote the current cell and neighbouring cell, respectively and Δt is the time-step calculated by applying the Courant-Friedrichs-Lewy (CFL) condition as follows,

$$\Delta t = Co\left(min_i \frac{A_i}{max_j(|v_i \cdot \mathbf{n}_{i,j}| + \sqrt{\phi_i}|\mathbf{n}_{i,j}|)}\right) \tag{3.28}$$

where $\phi = gh$, v is the velocity and Co is the safety factor, which is given a value of 0.9. However, for the INS solver, a more complex time integration method is employed, due to the use of pseudo-compressibility.

Pseudo-Compressibility Method

For the pseudo-compressibility method, the solution is obtained using a dual-time-stepping approach [68],[69]. To achieve this, a fictitious derivative with respect to artificial time must first be introduced to equation 3.1 to produce,

$$\frac{\partial}{\partial \tau}\int_{\Omega} \mathbf{U} d\Omega + \mathbf{I_0}\frac{\partial}{\partial t}\int_{\Omega} \mathbf{U} d\Omega + \oint_{S} \mathbf{F}\cdot\mathbf{n} dS = \int_{\Omega} \mathbf{H} d\Omega \tag{3.29}$$

$$I_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.30}$$

The finite volume discretisation of equation 3.29 is as follows,

$$A_i \frac{d\mathbf{U_i}}{d\tau} + A_i \mathbf{I_0} \frac{d\mathbf{U_i}}{dt} + \mathbf{C_i} = \mathbf{H_i} A_i \tag{3.31}$$

This is produced by integrating equation 3.29 over an arbitrary fixed cell, i, within the mesh. A_i is the area of cell i and C_i is the approximation of the flux integral. To simplify this into semi discrete form, the flux and source terms can be combined into a single residual, R_i^* as follows,

$$A_i \frac{d\mathbf{U_i}}{d\tau} + A_i \mathbf{I_0} \frac{d\mathbf{U_i}}{dt} + \mathbf{R_i^*} = 0 \tag{3.32}$$

At this stage, equation 3.32 needs to be discretised at a real time level n+1. Here, the implicit second order backward difference formula has been selected for the

real-time derivative,

$$A_i \frac{d\mathbf{U_i}}{d\tau} + A_i \mathbf{I_0} \frac{3\mathbf{U}_i^{n+1} - 4\mathbf{U}_i^n + \mathbf{U}_i^{n-1}}{2\Delta t} + \mathbf{R}_i^{*n+1} = 0 \tag{3.33}$$

where, Δt is the real time-step. This can be written in more compact form by combining the real-time discretisation with the residual, $\mathbf{R}_i^{*n+1}$, as follows,

$$A_i \frac{d\mathbf{U_i}}{d\tau} + \mathbf{R}_i^{n+1} = 0 \tag{3.34}$$

A time-accurate solution is found only once a steady-state solution has been found in artificial time at each real-time step. To achieve this, an explicit four stage Runge-Kutta scheme is implemented in pseudo-time [70]. Discretising equation 3.33 at artificial time level $m+1$ and rearranging gives,

$$\mathbf{U}_i^{n+1,m+1} = \mathbf{U}_i^{n+1,m} - \Delta\tau \frac{1}{A_i} (A_i \mathbf{I_0} \frac{3\mathbf{U}_i^{n+1,m} - 4\mathbf{U}_i^n + \mathbf{U}_i^{n-1}}{2\Delta t} + \mathbf{R}_i^{*n+1,m}) \tag{3.35}$$

The four stage Runge-Kutta scheme is now implemented as follows,

$$\begin{aligned}
\mathbf{U}_i^{n+1,0} &= \mathbf{U}_i^{n+1,m} \\
\mathbf{U}_i^{n+1,1} &= \mathbf{U}_i^{n+1,0} - \frac{\Delta t}{2A_i} \mathbf{R}_i^{n+1,0} \\
\mathbf{U}_i^{n+1,2} &= \mathbf{U}_i^{n+1,0} - \frac{\Delta t}{2A_i} \mathbf{R}_i^{n+1,1} \\
\mathbf{U}_i^{n+1,3} &= \mathbf{U}_i^{n+1,0} - \frac{\Delta t}{A_i} \mathbf{R}_i^{n+1,2} \\
\mathbf{U}_i^{n+1,4} &= \mathbf{U}_i^{n+1,0} - \frac{\Delta t}{6A_i} (\mathbf{R}_i^{n+1,0} + 2\mathbf{R}_i^{n+1,1} + 2\mathbf{R}_i^{n+1,2} + \mathbf{R}_i^{n+1,3}) \\
\mathbf{U}_i^{n+1,m+1} &= \mathbf{U}_i^{n+1,4}
\end{aligned} \tag{3.36}$$

The artificial time-step, $\Delta\tau$ is calculated from both the convective and diffusive components:

$$\frac{1}{\Delta\tau} = \frac{1}{\Delta\tau^C} + \frac{1}{\Delta\tau^D} \tag{3.37}$$

where the convective component is given by,

$$\Delta\tau^C = \frac{2A_iCo}{\sum_k(|\bar{\lambda}_k||\Delta S_k|)} \tag{3.38}$$

where Co is the safety factor, $\bar{\lambda}_k$ is the largest eigenvalue from Roes matrix and ΔS_k is the length of edge k. The diffusive component is expressed as,

$$\Delta\tau^D = \frac{Co(A_i)^2}{4/Re\sum_k(|\Delta S_k|^2)} \tag{3.39}$$

A limit should be applied to the artificial time-step. This is to prevent it from becoming too large in comparison to the real-time step and hence, prevent the discretisation from becoming unstable. For the implemented scheme, this limit is,

$$\Delta\tau = min(\Delta\tau, \frac{2}{3}\Delta t) \tag{3.40}$$

3.1.6 Unstructured Data Storage

For structured grids, data storage is very simple or even non-existent in many cases; using a Cartesian or Polar coordinate system, each dimension is defined by an index. However, due to the nature of unstructured grids, certain data must be stored. This data includes: locations of all grid points, locations of all cell centres, connections made between points to form cells and details of neighbouring cells. The current solver uses the following storage method:

1. Grid point locations are listed in two arrays named xpoint and ypoint,

$$xpoint = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ . \\ x_{np} \end{bmatrix} \quad ypoint = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ . \\ y_{np} \end{bmatrix} \tag{3.41}$$

   where np is the total number of grid points and the entries within the arrays are x and coordinates, respectively.

2. Cell centroid locations are defined similarly,

$$xcen = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ . \\ x_{nc} \end{bmatrix} \quad ycen = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ . \\ y_{nc} \end{bmatrix} \tag{3.42}$$

   where nc is the total number of cells and again, entries are x and y coordinates.

3. Details of the connections made between grid points to form the triangular

cells are given by a two-dimensional array,

$$connectivity = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ p_{3,1} & p_{3,2} & p_{3,3} \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ p_{nc,1} & p_{nc,2} & p_{nc,3} \end{bmatrix} \tag{3.43}$$

where each row of contains the three vertices forming the cell, given by an index from 1 to np. These must be listed in an anticlockwise manner to enable side vectors and outward pointing normals to be calculated in the correct directions.

4. Details of the neighbouring cells are stored similarly,

$$neighbours = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ c_{nc,1} & c_{nc,2} & c_{nc,3} \end{bmatrix} \tag{3.44}$$

where each row contains the three neighbouring cells given by an index from 1 to nc. Again these need to be listed in an anticlockwise direction.

Using this data the solver can then compute additional information required for the solution procedure. This includes cell side lengths, outward pointing normals and cell Areas. The cell areas are calculated as follows,

$$s = (s_1 + s_2 + s_3)/2$$
$$A = \sqrt{s(s - s_1)(s - s_2)(s - s_3)} \tag{3.45}$$

3.2 Unstructured Mesh Validation Test Results

3.2.1 Introduction

This section contains validation tests for the unstructured mesh solver. Firstly, the unstructured mesh generator, 'distmesh', was tested to ensure it is suitable for use with the solver. Secondly, benchmark tests were performed for both the shallow water equations (SWE) solver and the incompressible Navier-Stokes (INS) solver. The SWE solver was validated using a simple dam break test before before developing the code for use with the preferred INS equations. This new solver was validated with a lid driven cavity test, initially. This was chosen for the initial validation due to it's simplicity and the ease in which it can be compared to published numerical results. Next, the solver's ability to handle curved boundaries was tested with the flow past a stationary cylinder problem. Details of the numerical methods employed by both solvers can be found in Section 3.1

3.2.2 Unstructured Meshes

An essential property of the flow solvers is their ability to work with unstructured meshes. These meshes must be able to deal with complex geometries. It is for this reason that unstructured mesh generation is a very important part of this

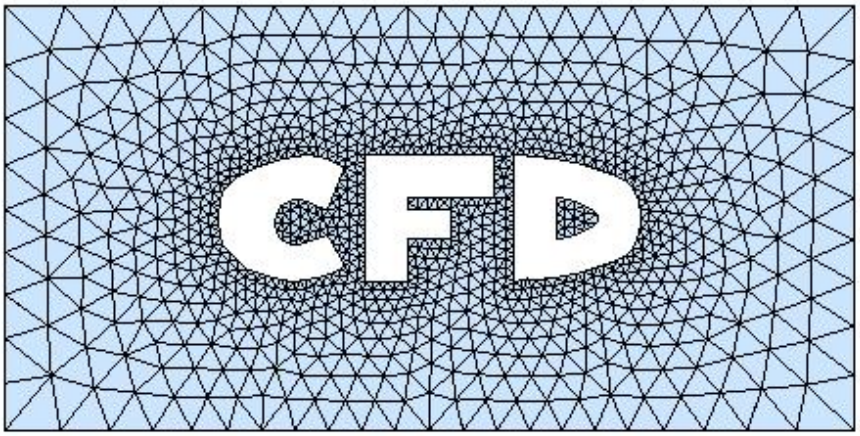


Figure 3.9: A non-uniform mesh around multiple complex geometries, generated using code in 3.11.

research project.

The function distmesh2d, was run to create non-uniform meshes for more complex geometries. The first demonstrates its ability to handle multiple complex geometries. A visual of this mesh is given in Figure 3.9 and the code used to generate this is displayed in Figure 3.11. The second is a more practical example. This mesh shown in Figure 3.10 is formed around an aerofoil. Each of the complex geometries within these meshes are defined by a set of coordinates. In Figure 3.11 these are given by objC, objF, objD1 and objD2. The aerofoil and the letters CFD are created using 85 and 102 fixed points respectively.

The non-uniform structure of the meshes demonstrates how a mesh can be constructed to conform to complex geometries. Directly surrounding the geometries are small triangles forming a fine grid. Moving away from the geometries, the triangles become larger, forming a coarser grid. This allows for more accurate solutions to be gained around the geometries or areas of most interest without having the computational expense of a completely fine mesh.

After testing the code by generating these meshes, it is clear that it can

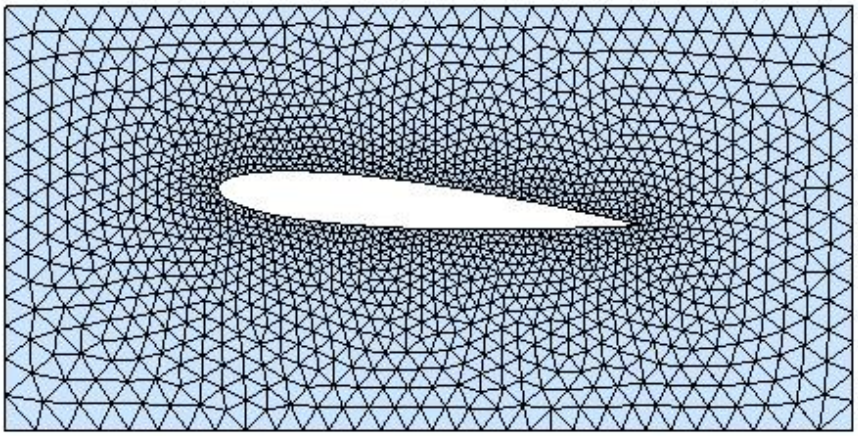


Figure 3.10: A non-uniform mesh around an aerofoil.

handle multiple complex geometries without loss of quality. The quality has been tested by calculating the skewness and aspect ratio for each mesh. The values for average skewness and aspect ratio respectively, for the first mesh are 0.17 and 1.06 and for the second mesh are 0.13 and 1.03. These results indicate very good quality meshes since the optimal values for skewness and aspect ratio are 0 and 1, respectively.

Using the signed distance functions to define the edge length provides very good results and allows the meshes to retain accuracy whilst reducing computational expense. The force-displacement function works well at keeping the boundaries intact. A very slight error does occur at the boundaries, but it appears small enough to avoid causing accuracy problems. Each of the meshes was generated by MATLAB in less than a minute and there appeared to be no issues maintaining the fixed points. Overall, the code performs very well and is simple to use. It produces high quality meshes, efficiently. It also has the ability to generate 3-dimensional meshes, but this has not been tested since there are no current plans to extend the solver's ability to 3-dimensional flow problems.

```
function d = cfd( p )

d1 = dpoly ( p, objC );
d2 = dpoly ( p, objF );
d3 = dpoly ( p, objD1 );
d4 = dpoly ( p, objD2 );

d5 = ddiff(d3, d4);

d6 = dunion ( d1, d2 );
d7 = dunion ( d6, d5 );

d8 = drectangle (p, 0.0, 2.0, 0.0, 1.0);

d = ddiff ( d8, d7 );

  return
end
.....

fd = @cfd;
fh=@(p)min(min(0.01+0.09*dpoly(p,objC),0.01+0.09*dpoly(p,objF))
,0.01+0.09*ddiff(dpoly(p,objD1),dpoly(p,objD2)));

box = [0.0,0.0; 2.0,1.0];

fixed = [0.0,0.0; 0.0,1.0; 2.0,0.0; 2.0,1.0; objC; objF; objD1;
objD2 ];

[ p, t ] = distmesh2d ( fd, fh, 0.02, box,  fixed );
```

Figure 3.11: Code used to produce mesh shown in Figure 3.9.

3.2.3 Circular Dam Break (SWE)

The initial Shallow Water equations solver has been tested on a circular dam break problem. This problem is based on the hypothetical test case studied by many researchers [71],[72],[73]. It is composed of a square domain measuring 200m × 200m with origin (0,0). Centred within this square is a circular dam with radius 50m. The initial water height within the dam is 10m and is 1m within the remainder of the square. This is illustrated by Figure 3.12.

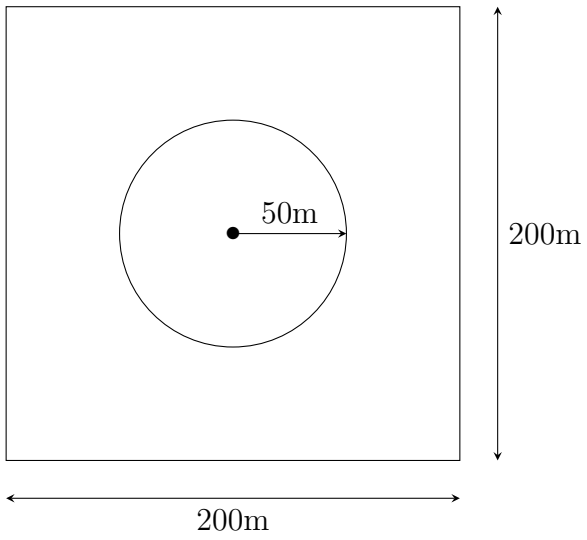


Figure 3.12: Setup of the circular dam break validation test.

It is assumed that the whole dam collapses to the ground instantaneously and no part of it obstructs the flow in any way. It is also assumed that the only force acting on the water is gravity. The initial velocity of the water in both the tank and the surrounding area is 0m/s and transmissive boundary conditions are implemented. This means that the water is free to flow through the boundaries without restriction.

An unstructured, uniform, triangular mesh generated using the distmesh function in MATLAB, covers the domain. Since the problem is symmetrical about

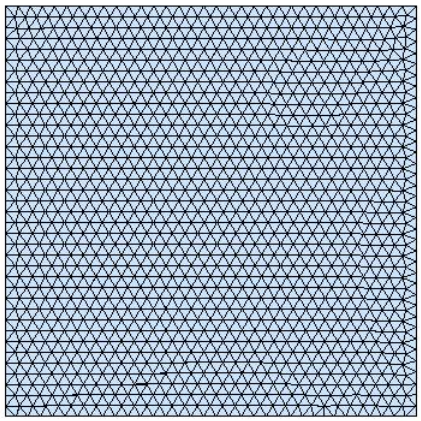


Figure 3.13: Unstructured mesh generated for the circular dam break test case.

both the x and y axis, for efficiency, a solution can be found for just one quarter of it. This solution can then be applied to the rest of the total domain using symmetry. Therefore, a mesh has been generated to cover one quarter of the full domain. This mesh is illustrated in Figure 3.13 and is constructed of 1,310 points connected to form 2,476 cells, giving a total of 9,904 cells covering all four quarters. The coarseness of the mesh was chosen for direct comparison to published results.

The water height, after running the program for 2.0 seconds is shown in Figure 3.14. From these results, it is clear to see that the cylindrical symmetry has not been maintained as it should. Instead, a hexagonal shape has been formed. This is caused by the use of a triangular mesh and illustrates the requirement for a mesh which conforms to the geometries within a problem. This problem has also been observed by Mingham and Causon [72] and was resolved by implementing a polar mesh.

Figure 3.15 shows a 3-dimensional view of the dam break. The water appears

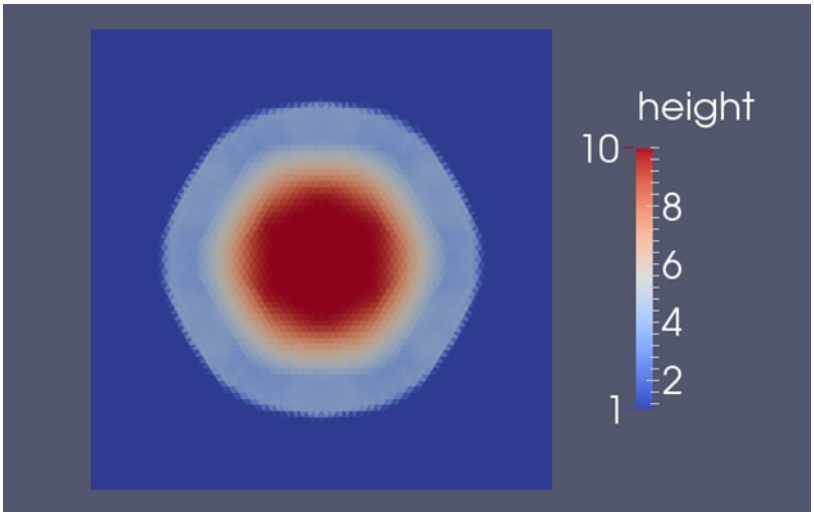


Figure 3.14: Height of water after 2 seconds of dam break.

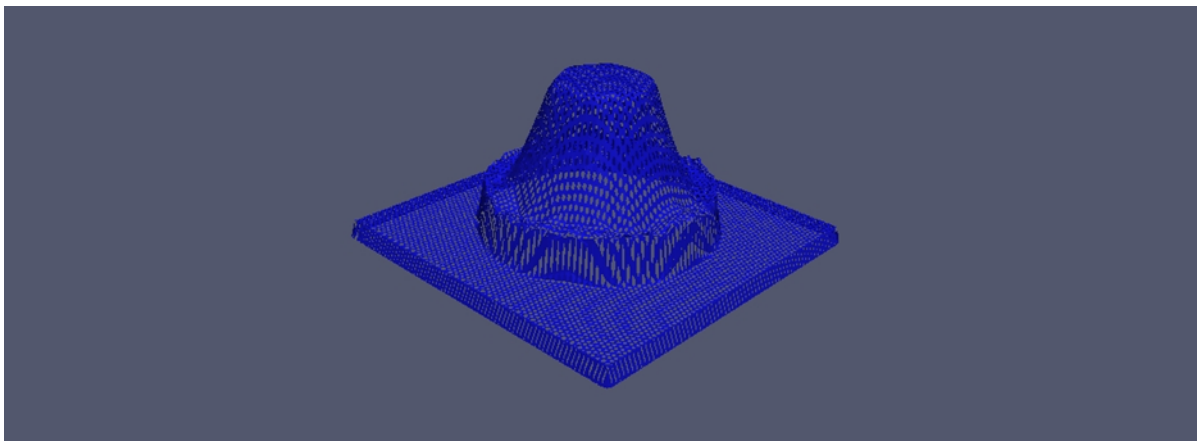


Figure 3.15: 3d view of dam break after 2 seconds.

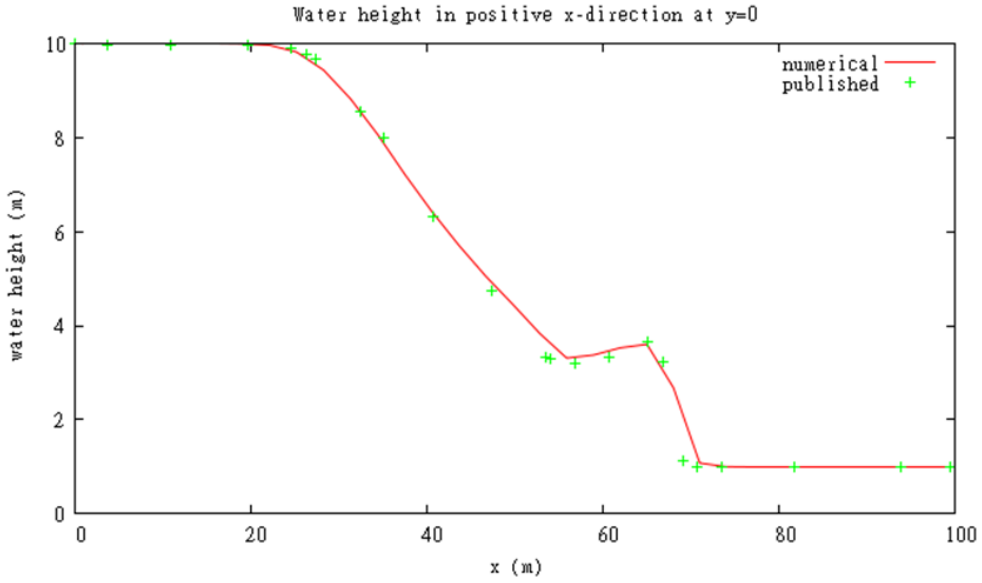


Figure 3.16: Water height along the cross section of the dam break, at $x > 0$ and $y = 0$ after 2 seconds.

to be acting as expected, with an advancing bore front and depression wave being formed. Figure 3.16 provides a closer look at the results. It shows the height profile along the positive x-axis at y=0. These results have been compared to published numerical results by Zoppou and Roberts [74] and are in good agreement. From this comparison of results, it can be concluded that the program is functioning as desired, signifying that there are no errors present within it. This solver can now be developed into an incompressible Navier-stokes solver.

3.2.4 Lid Driven Cavity (INS)

The Incompressible Navier-Stokes solver was first tested using the lid driven cavity problem, which consists of a square domain with side length, L=1. The square, as illustrated in Figure 3.17, represents a tank full of water with solid, stationary walls, but a moving lid. The lid is moving at a constant speed in the x-direction

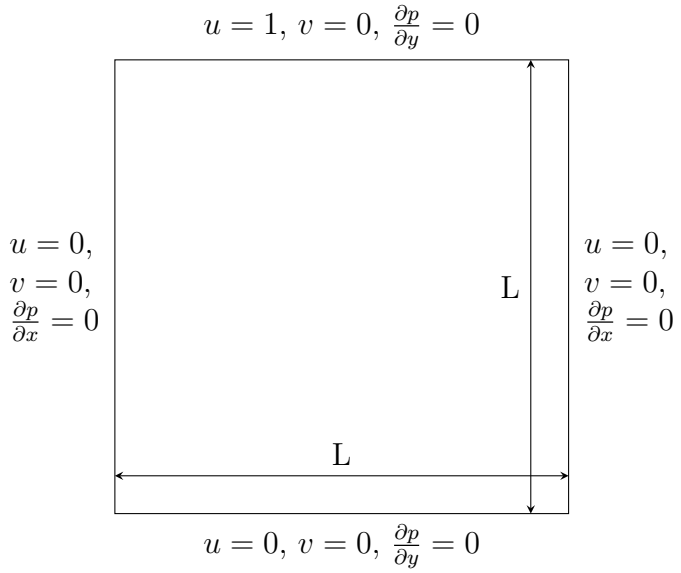
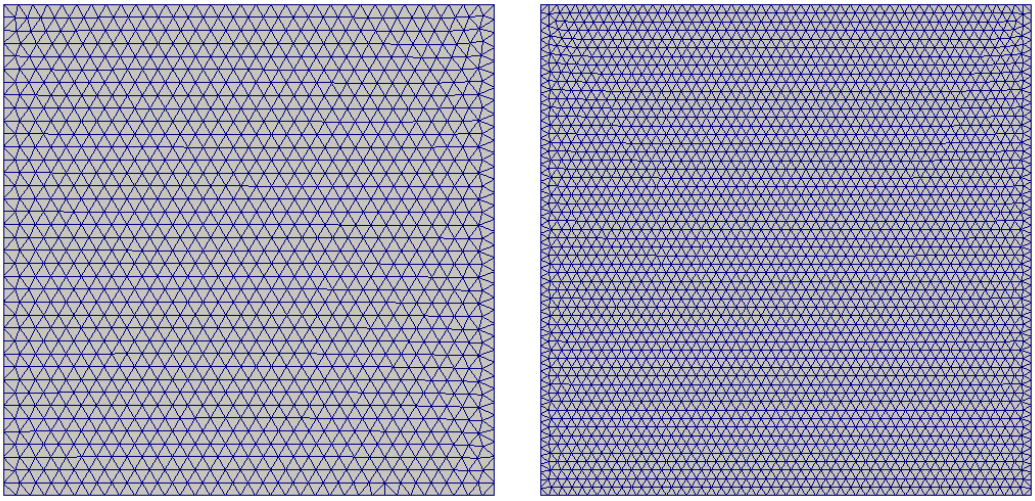


Figure 3.17: Setup of lid-driven cavity test.

of u=1. There is a zero pressure gradient at the boundaries in the normal direction and initially the water velocity and pressure within the tank is zero. These parameters were chosen to match those used in published studies [75],[76],[77].

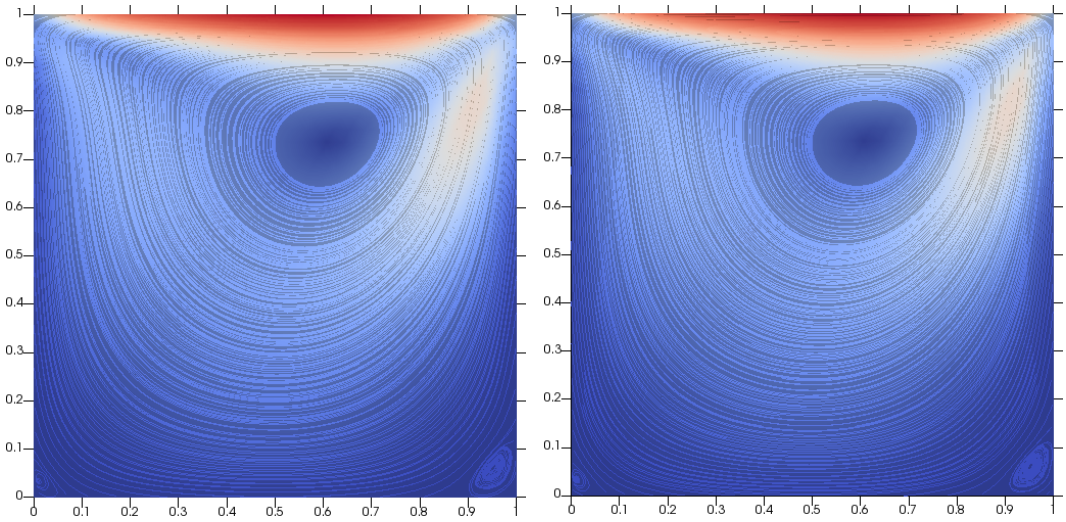
This test case was run for the Reynolds numbers, Re = 100, 400 & 1000. For all cases, the pseudo-compressibility factor, β was set to 10 for faster convergence. Artificial time-steps were performed until the relative error was less than 10^{-5} within each real time-step. For Re=100, a real time-step, Δt of 0.1 was found to be sufficient and the simulation was run on a uniform mesh with a total of 2,476 cells, named mesh 1. However, it was also performed on a finer mesh, named mesh 2, with a total of 5,645 cells to ensure the accuracy of the coarser mesh. Both of these meshes are illustrated by Figure 3.18.

It should be noted that to obtain a steady state solution with the finer mesh, Δt needed to be reduced to 0.05. Figure 3.19 shows the velocity streamlines of the steady state solution obtained on both meshes, where a large primary vortex has formed along with smaller secondary vortices in the lower corners. There is



(a) Mesh 1: 2,476 Cells (b) Mesh 2: 5,645 Cells

Figure 3.18: The meshes used for the lid driven cavity tests.

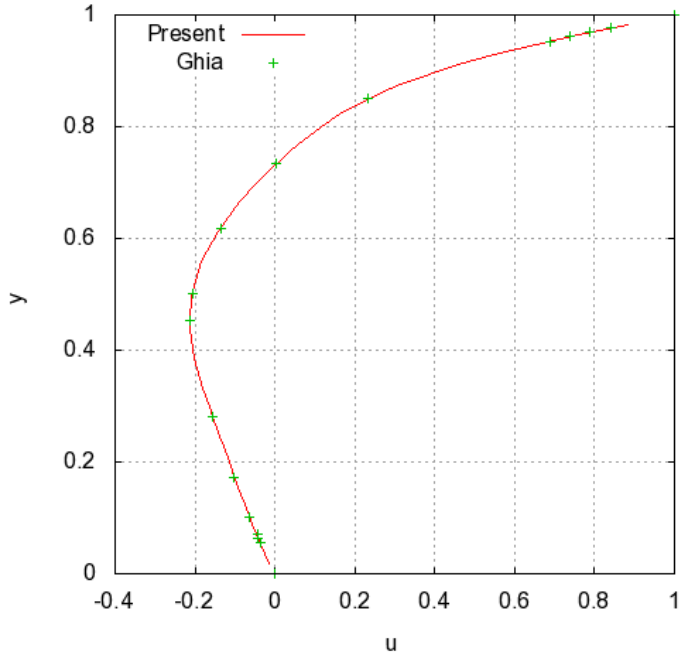


(a) 2,476 Cells (b) 5,645 Cells

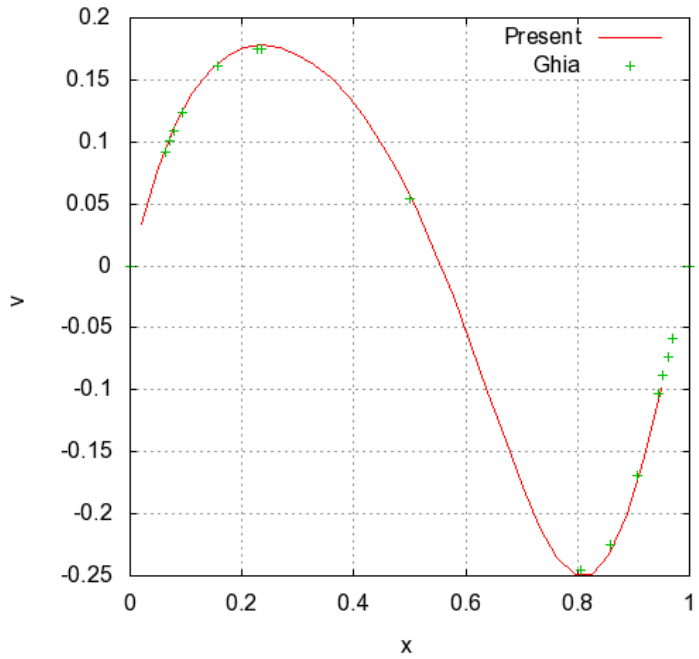
Figure 3.19: Velocity streamlines for steady state solutions on both meshes (Re=100).

no noticeable difference between the solutions on the two meshes, indicating that the coarser of the two is suitable in this case. Here the solver has demonstrated its ability to capture small flow features on a relatively coarse mesh. Figure 3.20 shows plots of the velocity profiles through the geometric centre-lines of the domain with a comparison to the numerical results provided by Ghia et al. [75],

which are in very good agreement.

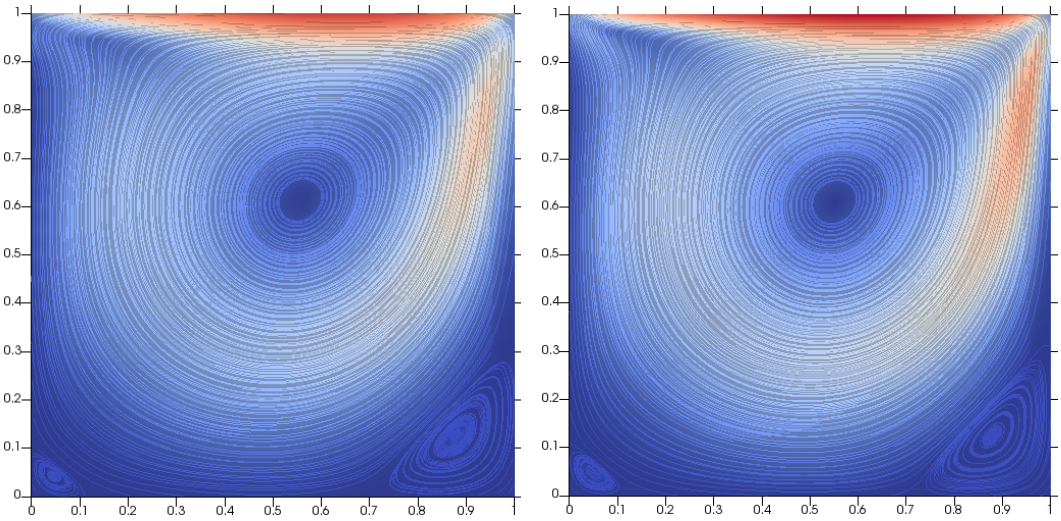


(a) u-velocity profile



(b) v-velocity profile

Figure 3.20: Velocity profiles through geometric centre-line of cavity (Re=100).



(a) Mesh 1: 2,476 cells (b) Mesh 2: 5,645 cells

Figure 3.21: Velocity streamlines for steady state solutions on meshes 1 & 2 (Re=400).

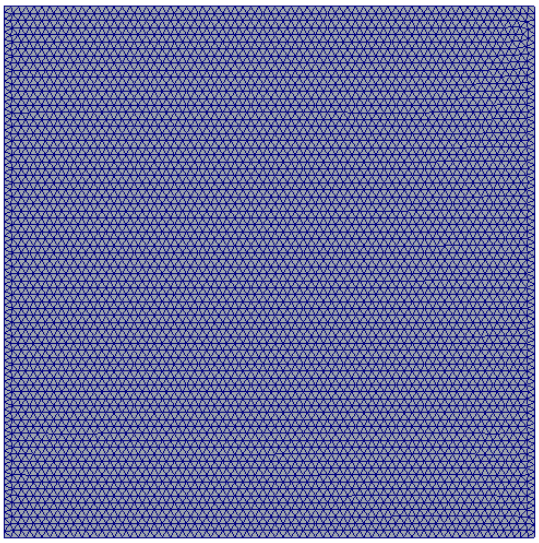


Figure 3.22: Mesh 3: 10,035 cells.

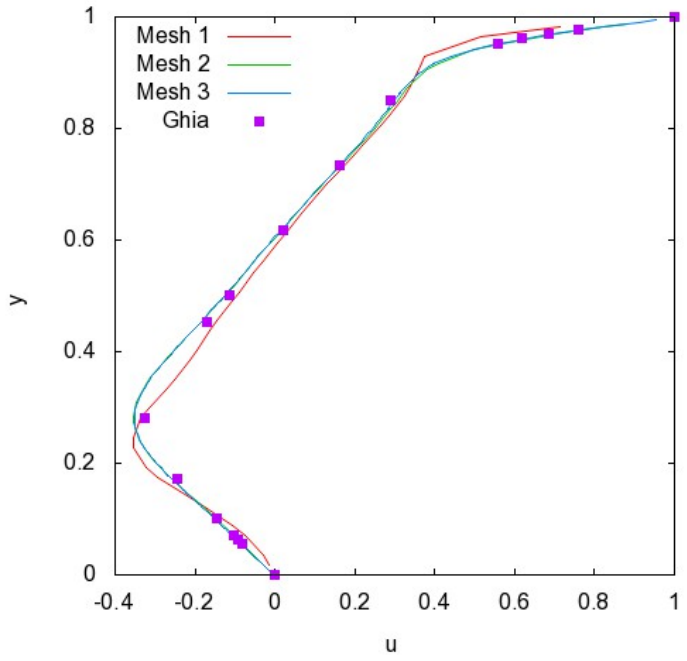
For Re=400, solutions were found using meshes 1 & 2. A comparison of velocity streamlines obtained by mesh 1 and mesh 2, given in Figure 3.21 shows very little difference. Again, mesh 1 has shown capability in capturing smaller secondary vortices. However, when comparing the velocity profiles obtained through the geometric centre-lines, a difference was apparent. For this reason, the mesh

was refined again and solutions were obtained on all three for both Re=400 and Re=1000. The new mesh, illustrated by Figure 3.22 had a total of 10,035 cells and was named mesh 3. The velocity profiles for all three meshes and numerical results provided by Ghia et al. [75] are given by Figures 3.23 and 3.24 for Re=400 and Re=1000, respectively. For each of the cases, the solutions provided by mesh 2 and mesh 3 are in good agreement, implying that a mesh independent solution has been obtained with mesh 2 in both cases. These solutions are both in agreement with the published numerical results included. Magnified views of the secondary vortices for Re=400 obtained with mesh 2 are provided by Figure 3.25. Velocity streamlines of the steady state solution obtained by mesh 2 for Re=1000 is given by Figure 3.26. The sizes and locations of the vortices (both primary and secondary) in each case are also very comparable with those published by Ghia et al [75].

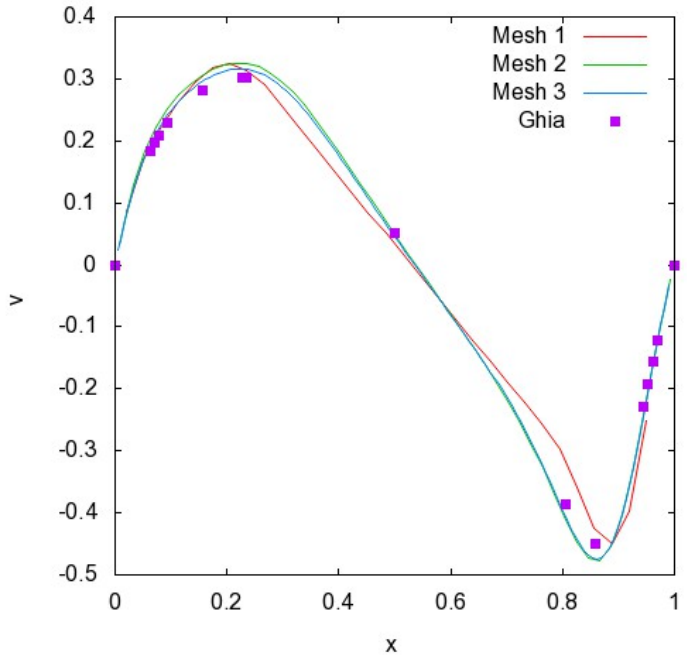
3.2.5 Flow Past a Stationary Cylinder

The solver has also been tested for the flow past a stationary cylinder problem, which have been studied extensively during the past few decades [78],[79]. This simulation is useful to test the solvers ability to handle curved boundaries. The 2-dimensional problem, illustrated by diagram 3.27 consists of a circle with a diameter, D, of 1, situated at the centre of a square domain of size 30D $\times$ 30D. This large domain has been chosen in order to reduce effects from the walls. The left boundary has a constant velocity of 1 in the x-direction and at the upper and lower boundaries, a slip condition is applied. The circle has solid wall boundary conditions with a zero normal pressure gradient.

The flow problem has been run for Reynolds numbers (Re) of 20, 40, 100 and 185. The lower Re values of 20 and 40 have been chosen to simulate steady state

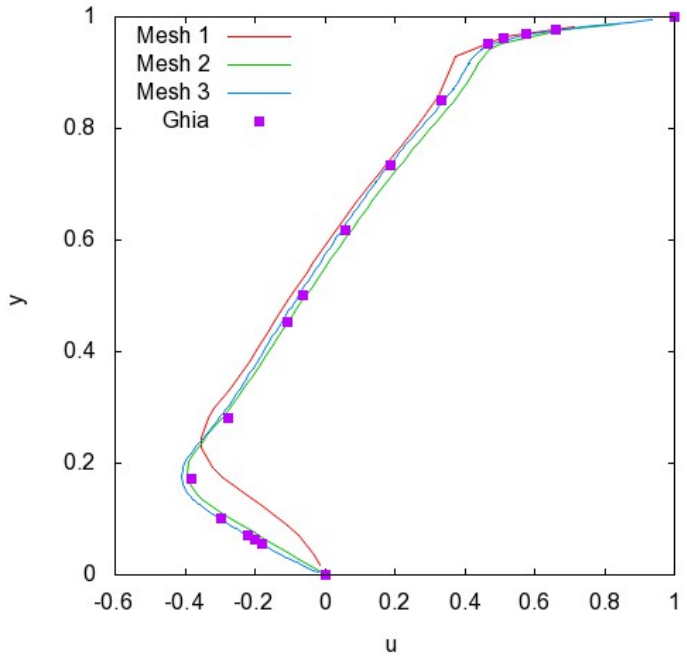


(a) u-velocity profile

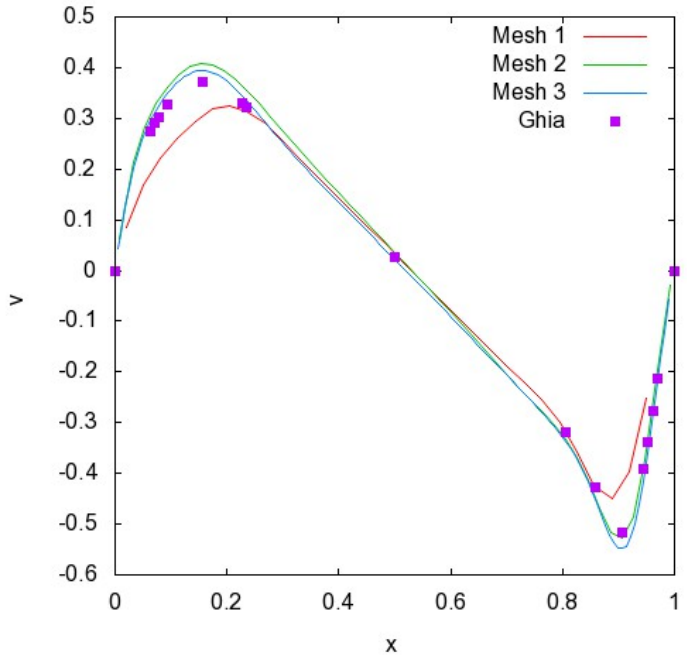


(b) v-velocity profile

Figure 3.23: Velocity profiles through geometric centre-line of cavity (Re=400).

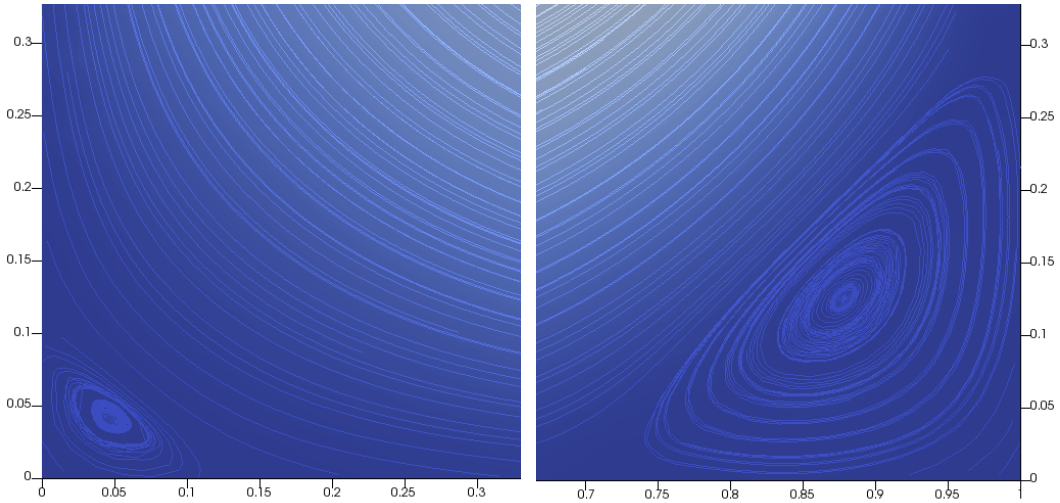


(a) u-velocity profile



(b) v-velocity profile

Figure 3.24: Velocity profiles through geometric centre-line of cavity (Re=1000).



(a) Left (b) Right

Figure 3.25: Velocity streamlines for secondary vortices in lower corners (Re=400).

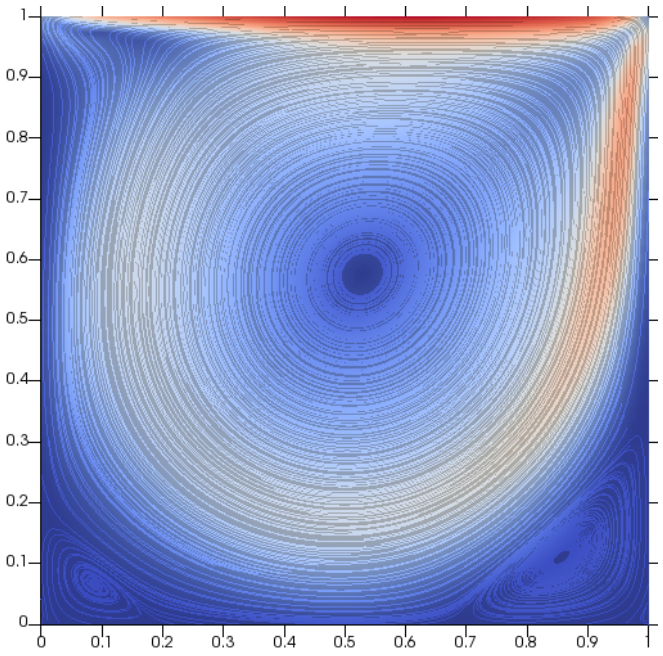


Figure 3.26: Velocity streamlines for steady state solution (Re=1000).

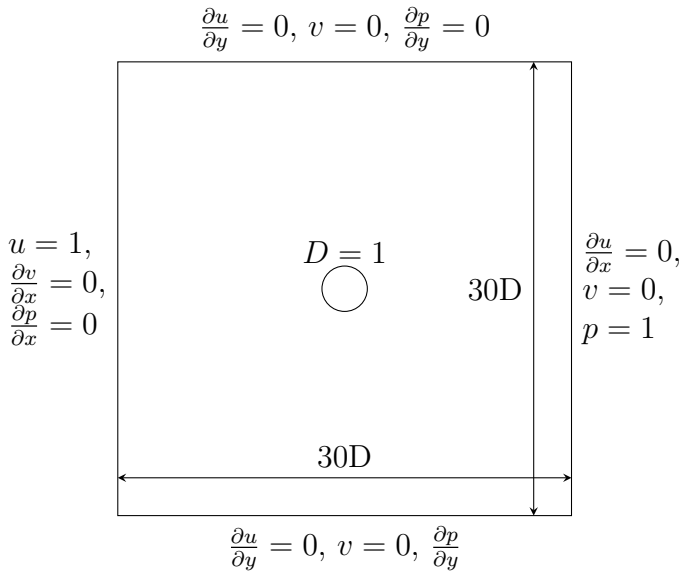


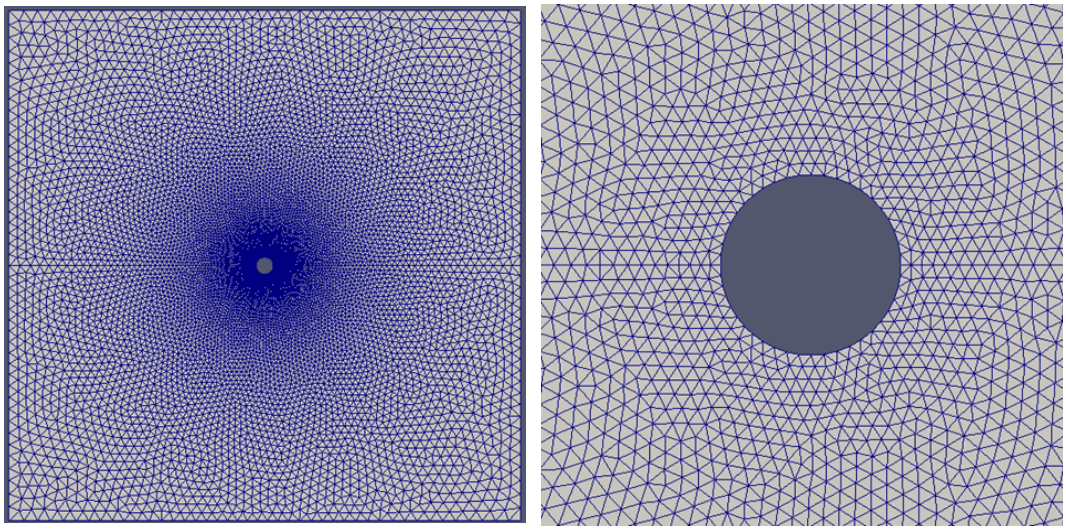
Figure 3.27: Setup of flow past a stationary cylinder test case.

flow and the higher values of 100 and 185 have been chosen so that unsteady vortex shedding can be observed. These values are all regularly used for this test case, meaning there are many published results for comparisons to be made with. Furthermore, a Reynolds number of 185 has been extensively studied within published literature for an oscillating cylinder case. In order to run an oscillating cylinder case, the natural shedding frequency of the stationary case is required, making this test essential. Due to the importance of this Reynolds number in future simulations, it has been chosen as the value to be used at every development stage of the solver in order to make comparisons. The meshes for each of these simulations were chosen through mesh convergence tests. The results of these mesh convergence tests are given in Table 3.1. All meshes have a non-uniform distribution, determined by a non-uniform factor, which defines the extent to which the mesh will be stretched from the cylinder, outwards. For this test case, a non-uniform factor of 0.5 has been applied throughout. This was found to give a reasonable increase in neighbouring cell sizes, whilst ensuring cells on

$Min\Delta S$	No.Cells	Re=20		Re=40		Re=100		Re=185	
		C_D	L_w	C_D	L_w	$\bar{C}_D$	St	$\bar{C}_D$	St
0.08	13,810	2.02	0.89	1.52	2.19	1.372	0.160	1.400	0.189
0.07	18,010	2.03	0.90	1.53	2.21	1.374	0.162	1.395	0.189
0.06	24,788	2.03	0.90	1.53	2.21	1.354	0.162	1.380	0.189
0.05	35,516	-	-	-	-	1.353	0.162	1.379	0.189

Table 3.1: Drag coefficients and Strouhal numbers for mesh convergence tests for different Reynolds numbers.

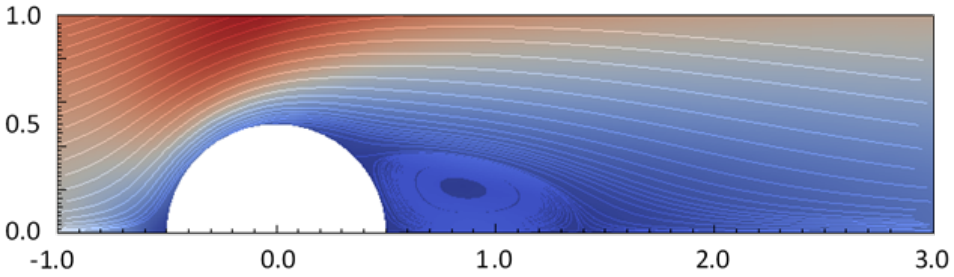
the boundaries are not too large. All of these simulations were run with a real time-step (dt) of 0.1 and a pseudo-compressibility factor (beta) of 1. This beta value was chosen through experimentation and was found to produce the best convergence.



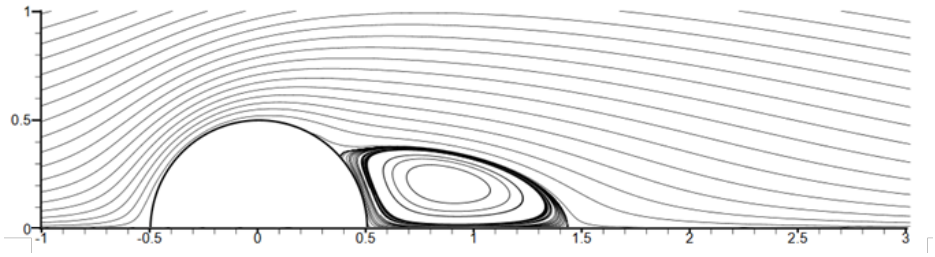
(a) Full Mesh (b) Cells around cylinder

Figure 3.28: The mesh used to obtain a mesh independent solution.

For the case where Re=20 and 40, the mesh converged solution was found to have a total of 18,010 cells, with a smallest cell side (around the cylinder) of 0.07. Figure 3.28 shows the mesh, with a close up of the cells around the cylinder. A steady state solution was reached in both cases, with two symmetrical vortices

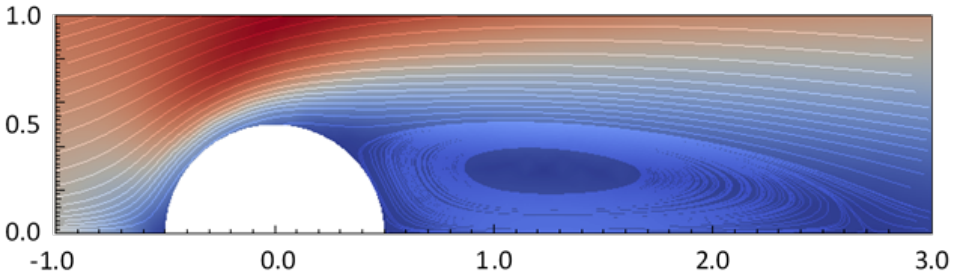


(a) Present

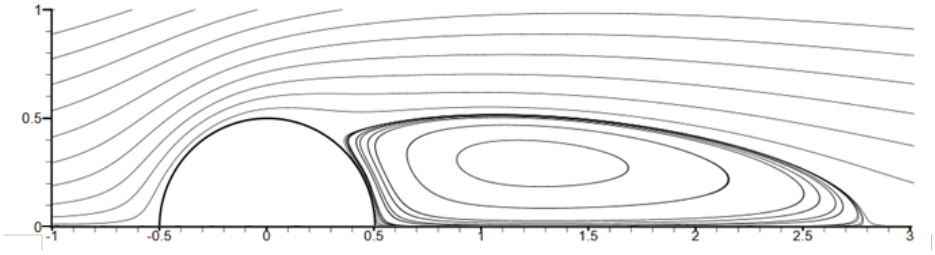


(b) Ma et al. [76]

Figure 3.29: Velocity streamlines of flow past a stationary cylinder (Re=20).



(a) Present

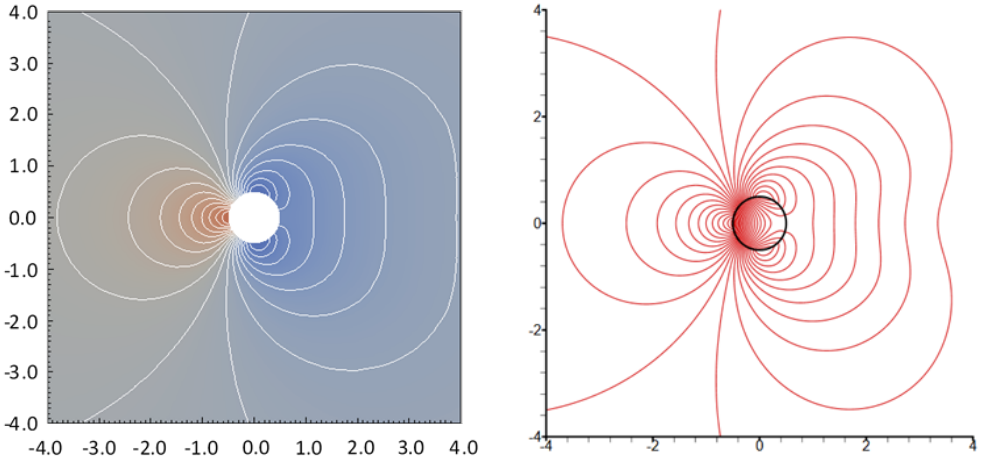


(b) Ma et al. [76]

Figure 3.30: Velocity streamlines of flow past a stationary cylinder (Re=40).

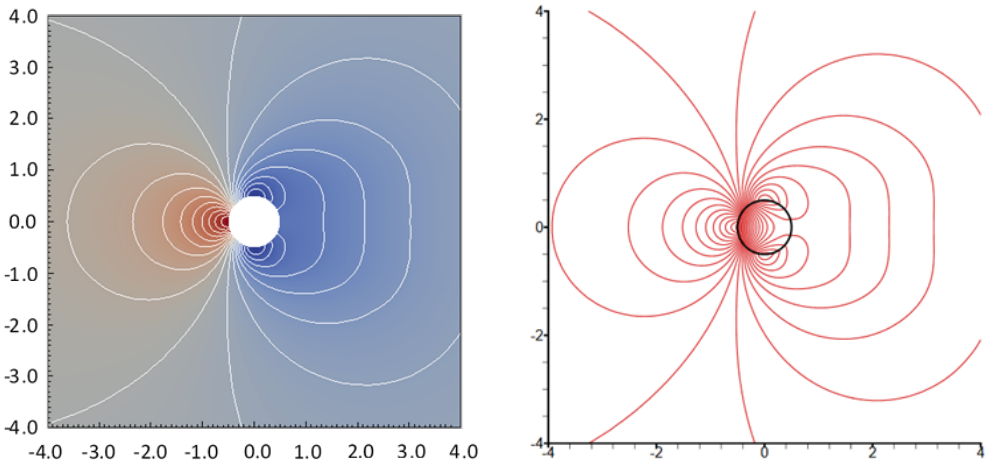
forming behind the cylinder. For Re=20, the drag coefficient (C_D) was found to be 2.03 with the length of the re-circulation zone (L_w) at 0.90. For Re=40, C_D was found to be 1.53 with L_w at 2.21. Figures 3.29 & 3.30 show the velocity

streamlines for the present results compared with published numerical results for Re=20 and 40, respectively. In each case, the recirculation length, L_w, can clearly be depicted. Figures 3.31 and 3.32 show the pressure around the cylinder for the present results compared with numerical published results for Re=20 and 40, respectively.



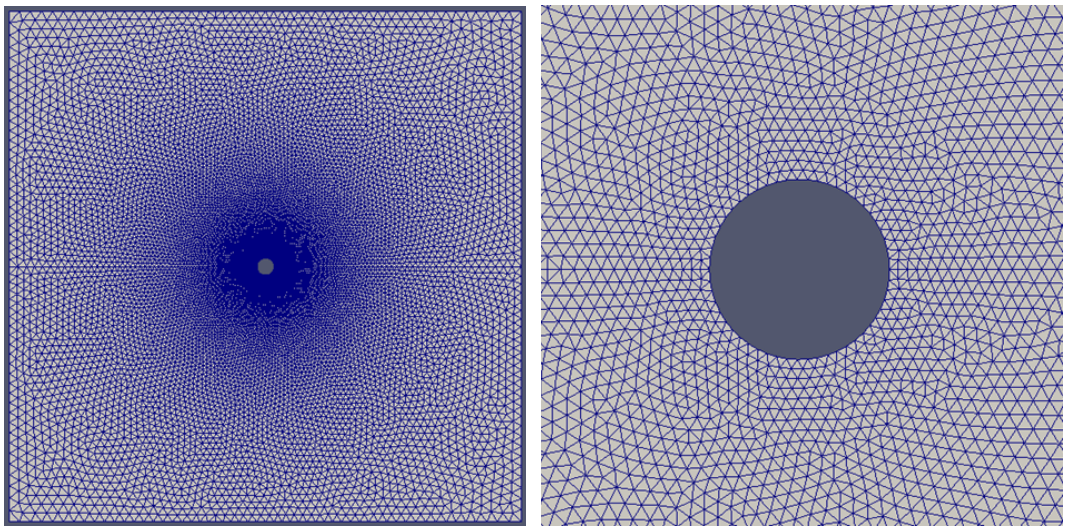
(a) Present (b) Ma et al. [76]

Figure 3.31: Pressure contours for flow past a stationary cylinder (Re=20).



(a) Present (b) Ma et al. [76]

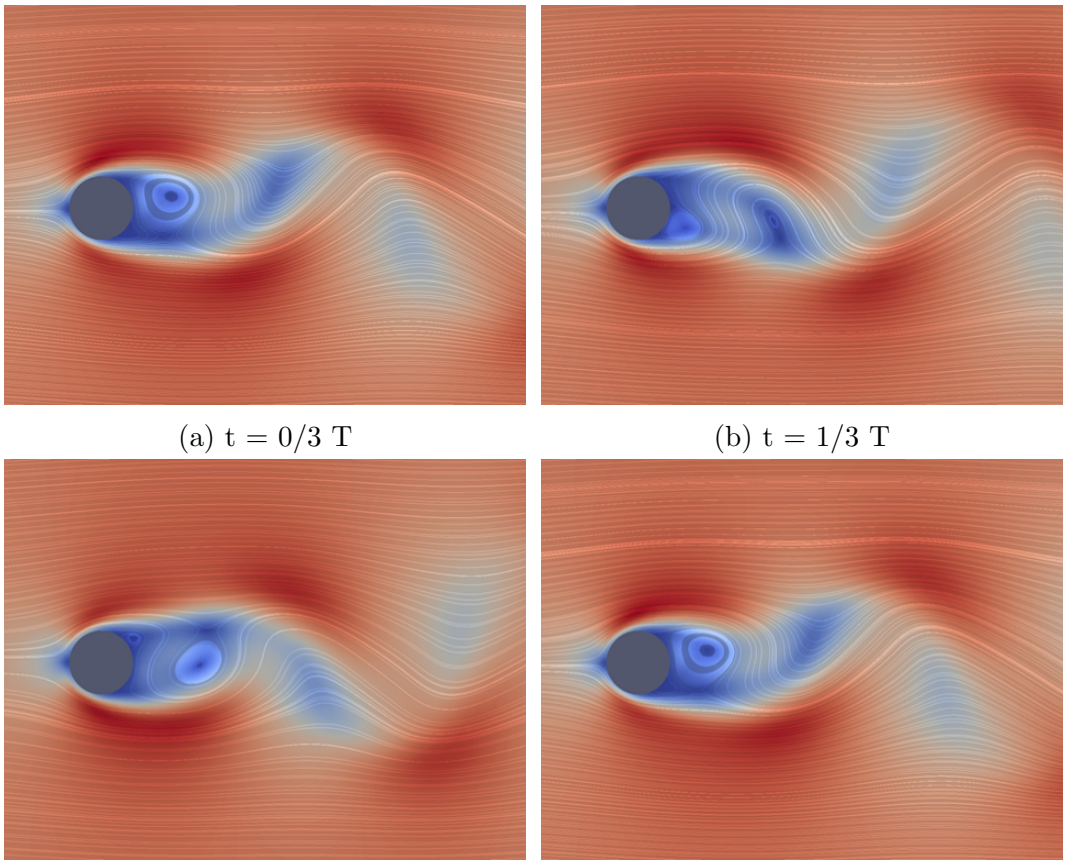
Figure 3.32: Pressure contours for flow past a stationary cylinder (Re=40).



(a) Full Mesh (b) Cells around cylinder

Figure 3.33: The mesh used to obtain a mesh independent solution.

For Re=100 a finer mesh was required to provide a mesh independent solution. This mesh consists of 24,788 cells in total, with a smallest cell side of 0.06. Figure 3.33 shows the mesh, with a close up of the cells around the cylinder. It is known that for Re>40, a steady state solution cannot be reached. Instead, vortices form behind the cylinder and shed in a periodic fashion, forming oscillations in the flow behind the cylinder. This can clearly be seen in Figure 3.34, which shows the velocity streamlines for the times, 0/3T, 1/3T, 2,3T and 3/3T, where T is period of vortex shedding. When vortex shedding occurs, the drag and lift coefficients oscillate in time. This can be seen in Figure 3.35, showing the lift and drag coefficients over time. The Strouhal number, St is the name given to a dimensionless number describing the rate of vortex shedding. It is calculated using $St = fD/V$, where f is the frequency of oscillations, D is the diameter of the cylinder and V is the velocity. In this case, since the velocity and diameter are both 1, it simplifies to $St = f$. For the present simulation, this is found to be 0.162 with an average drag coefficient, $\bar{C_D}$, of 1.354.



(a) t = 0/3 T (b) t = 1/3 T

(c) t = 2/3 T (d) t = 3/3 T

Figure 3.34: Velocity streamlines for a period, T, of vortex shedding.

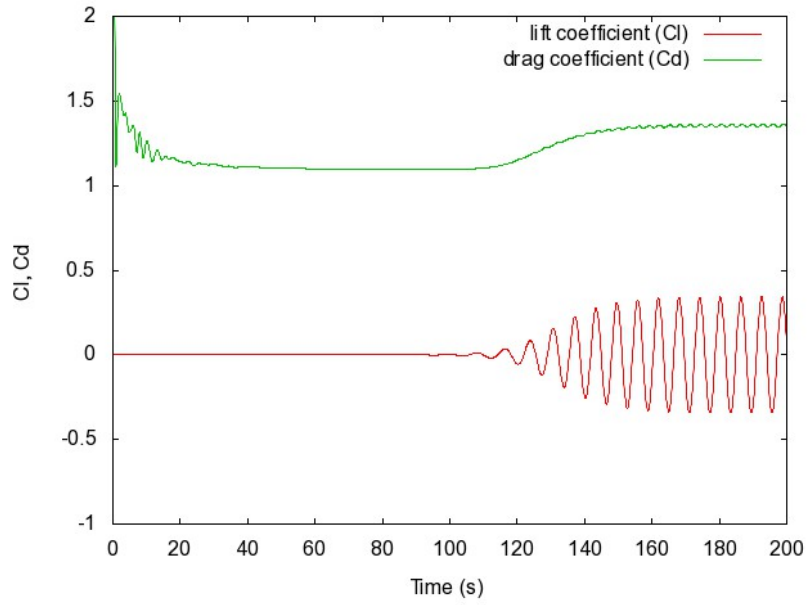


Figure 3.35: Plot of the drag and lift coefficients over time (Re=100).

For Re=185, a converged solution was on the same mesh as for Re=100. $\bar{C_D}$ and St were calculated as 1.38 and 0.189, respectively. A plot of the drag and lift coefficients over time is shown by Figure 3.36 and the velocity streamlines at t=200 is given in Figure 3.37.

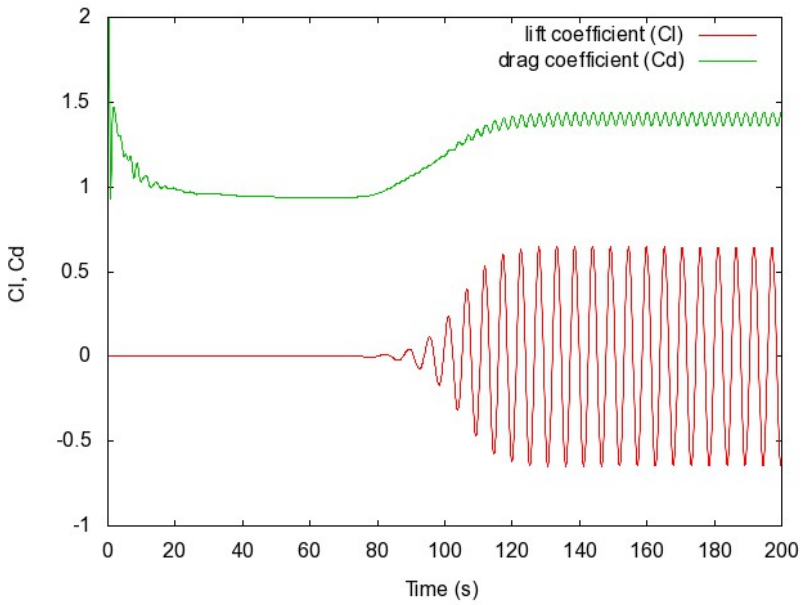


Figure 3.36: Plot of the drag and lift coefficients over time (Re=185).

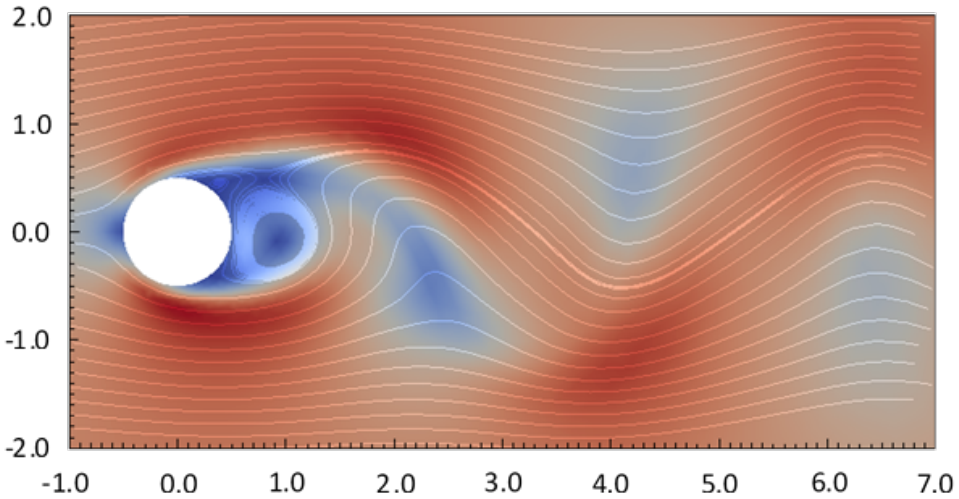


Figure 3.37: Velocity streamlines of flow past stationary cylinder (Re=185).

	Re=20		Re=40		Re=100		Re=185	
	C_D	L_w	C_D	L_w	$\bar{C}_D$	St	$\bar{C}_D$	St
Ma et al. [76]	2.08	0.92	1.56	2.32	1.410	0.167	-	-
Chung [80]	2.05	0.96	1.54	2.30	1.392	0.172	1.375	0.184
Fornberg [81]	2.00	0.91	1.5	2.27	-	-	-	-
Hartmann [82]	2.043	0.972	1.535	2.240	1.358	0.164	-	-
Lu & Dalton [83]	-	-	-	-	-	-	1.31	0.195
Present	2.03	0.90	1.53	2.14	1.354	0.164	-	-

Table 3.2: Drag coefficients and re-circulation lengths / Strouhal numbers for present solver compared with published results for Re=20, 40, 100 & 185.

Table 3.2 gives a comparison of the present results with published numerical results for Re = 20, 40, 100 & 180. These results consist of the drag coefficients for each case and the Re-circulation Length or Strouhal number, as appropriate.

3.2.6 Summary

Validation tests have been carried out for the unstructured mesh solver. Firstly, a simple dam break test was simulated using the Shallow Water equations. This provided sufficient results in comparison to published numerical results to conclude that the solver was working. Since these equations were employed simply as a starting point for the code development, no further test cases were run and development began with the preferred incompressible Navier-Stokes equations (INS).

The INS solver was initially validated using a standard lid-driven cavity test. This provided very good results in comparison to published numerical results for the same test [75]. It was found that a good solution could be found for a low Reynolds number (Re) of 100 using a relatively coarse mesh of 2,476 cells and large

real time-step of 0.1. This includes the velocity profiles through the geometric centre-lines as well as well as the capability to capture smaller secondary vortices in the corners of the cavity. Upon increasing this Re value to 400 and 1000, it was found that both a finer mesh of 5,645 cells and smaller time-step of 0.05 were required to gain a solution in good agreement with the aforementioned published results. For all Re values, $\beta = 10$ was found to provide the best convergence. Overall, for this simple test case the solver was found to accurately model the viscous flow.

Next, flow past a circular cylinder was tested. Again, various Re values were used to provide a range of results for comparison with selected published numerical results [76],[80],[81],[82],[83]. For the lower Re cases (Re= 20 & 40), the drag coefficient and re-circulation length (of the vortex formed behind the cylinder) were found to be within the range of values given by these published results. A mesh converged solution was obtained on a mesh with a total of 18,010 cells and minimum cell side length of 0.07. For the higher Re cases cases (Re = 100 & 185) a finer mesh of 24,788 cells and a minimum cell side length of 0.06 was required for a mesh converged solution. The average drag coefficient and Strouhal number were again found to be within the range of the specified published results. For all of these cases a real time-step of 0.1 was used and a β value of 1 was found to give the best convergence. The results obtained suggest the code is very capable of dealing with curved boundaries. Although sufficient accuracy has been obtained, it may have been beneficial to reduce the time-step. However, this solver has been found to be far too slow, making a decrease in real time-steps not feasible. Factors contributing to this large run-time include the unstructured form of the data (see Section 3.1.6), the large domain in which the mesh must cover and the choice of explicit artificial time marching. Implicit artificial time marching has been avoided because it requires the use of complex sparse matrices due to

the unstructured form of the mesh. Although this seem like it is a considerable problem at this stage, it should be noted that this code has been developed to be used for overlapping grids, rather than for a whole domain. When used for this purpose, timing problems should not be of a concern. This is because the area in which they will cover will be dramatically reduced, with the bulk of the domain being covered by a structured grid. For this reason, it was decided to disregard this as an issue at this stage and observe how the code performs on overset grids.

Chapter 4

Overset Grids Solver

4.1 Overset Grids Numerical Methods

4.1.1 Introduction

The unstructured grid solver has been extended for overset capability. This overset capability allows for unstructured meshes to be generated around solid bodies, which then overlap a simple Cartesian background grid. Using unstructured meshes for the bodies allows for complex geometries to be well represented. The use of separate, overlapping grids means that bodies can move arbitrarily whilst communications are held with the background mesh.

Beneath an overlapping mesh will lie a part of the background mesh that needs to be removed from the solution domain. This is for two reasons - Firstly, the solution is already being calculated for the part of the flow domain on the mesh above. But secondly, and more importantly, the mesh above may be around a solid object, meaning that the background mesh will not be correctly representing the domain. The process of removing this part of the mesh from the solution domain is called hole-cutting.

Since the background mesh and the overlapping mesh are of Cartesian and unstructured forms, respectively, separate flow solvers are required for each of these. This is because data is stored very differently for each type of mesh (see Section 3.1.6). The unstructured mesh uses the solver discussed in Chapter 3. The background mesh uses the CMMFAs in house solver for Cartesian meshes, AMAZON-SC. These solvers can be combined easily since they use the same methods; they are incompressible Navier-Stokes solvers, using a cell-centred finite volume method and Roes flux function. The combined codes have been developed for overset capability, which required some changes to be made to the in-house code. A dual time-stepping approach is enforced. For each artificial time-step, both solvers are run and the solution is exchanged between the two by grid interpolation.

4.1.2 Hole-Cutting and Grid Interpolation

For the current solver, a very simple hole-cutting method has been applied. This method has many limitations but has been utilised in order to gain a working overset solver. It will then be adapted and improved.

Firstly, the process will only work if a uniform Cartesian background mesh is in use and if the overlapping mesh is rectangular. It takes the location of each side of the overlapping mesh and uses this to find i or$j$ index of the background cells in which the side lies. It then gives a maximum and minimum i and j index for the hole-cut to be made based on leaving an overlap that contains two whole background cells (see Figure 4.1). The number of whole cells for the overlap has been chosen through experimentation. During these experiments, it was found that having an overlap of less than two whole cells resulted in a change in the solution. However, increasing the overlap to more than two whole cells had no

effect on the solution. For efficiency, the minimum requirement was used. Because of the simple nature of this method, the hole will always be rectangular with sides parallel to the x and y axis. As a result, it does not allow for any rotation of overlapping meshes.

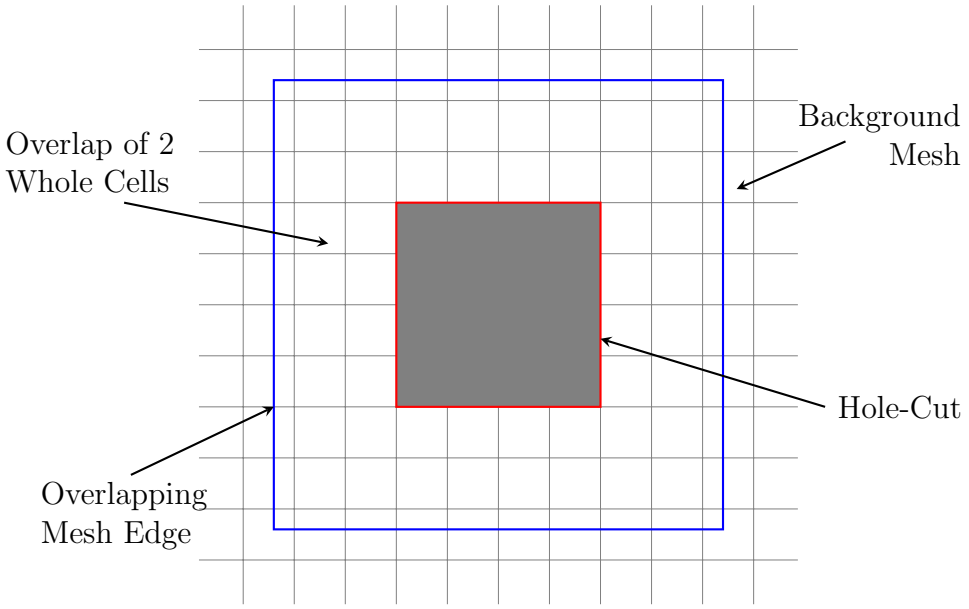


Figure 4.1: Diagram of overset grids, showing hole-cut on background grid.

Once the hole has been cut, the next task is to find the cells on each grid that will be used for grid interpolation. The method implemented is a simple linear interpolation method as proposed by Kang et al. [84]. The cells receiving the data are ghost cells. For the background mesh, these are cells reflected along the lines of the hole-cut. For the overlapping mesh, these are cells reflected along the exterior boundaries of the mesh. Solution data is interpolated to the ghost cells on the overlapping mesh from donor cells on the background mesh. These donor cells are found simply by taking the location of the centre of them and finding the background cell that it lies in. As with the hole-cutting, this method requires the background mesh to be uniformly distributed. Solution data is interpolated to the background mesh from donor cells on the overlapping mesh. Finding these cells

requires a distance search algorithm. For each ghost cell, the distance between its centre and the centre of each of the cells on the overlapping mesh is found. The cell whose centre is the minimum distance from the ghost cell is stored as the donor cell. This can be a time consuming process, especially if the overlapping mesh is moving, as this process would need to be repeated for every time-step.

Grid interpolation is the process in which solution data is transferred between meshes. This is performed by interpolating solution data from donor cells to ghost cells. U_{ghost} is the interpolated solution, calculated as follows,

$$U_{ghost} = U_{donor} + \nabla U_{donor} \cdot DG \tag{4.1}$$

where, U_{donor} is solution stored at the centre of the donor cell, ∇U_{donor} is the cell gradient and DG is the distance vector from the donor cell to the ghost cell.

4.1.3 Implicit/Explicit Hybrid

Upon running a validation test for the overset solver at its current development stage, it was discovered to be inefficient in terms of the computational time required for a complete simulation. This problem was observed with the unstructured solver, but as discussed in Section 3.2.6, it was thought that this would become negligible once the unstructured solver was covering only a small portion of the total domain in the overset approach. A likely cause of the problem is the use of explicit 4-stage Runge-Kutta time marching, due to the multiple stages, costing CPU time and overall slow convergence of explicit techniques. Excluding the multiple stages and instead employing a simple first order explicit scheme did not improve the CPU time. Instead, the solution lost stability and resulted in the program crashing with no results obtainable.

A solution has been found by introducing a new hybrid of implicit and explicit

artificial time integration methods. An implicit time integration method is applied to the Cartesian background mesh. Implicit methods compute the solution over the whole mesh rather than on an individual cell basis. Also, the method is less restrictive on time-stepping and requires far less artificial time-steps to converge. As a result of these factors, it is a much faster method. However, an explicit time integration method is maintained for overlapping unstructured meshes. This means that the use of complicated sparse matrices, that would be required for an implicit method on unstructured grids, can be avoided. Also, implicit methods are difficult to parallelise and this process would be further complicated by the unstructured form of the data [8],[85]. Parallelisation of the solver would allow for much larger real-world problems to be simulated, since the computational expense could be managed between multiple processors. Although parallelisation is not currently within the aims of this research, it is worth considering this as a future possibility. For each artificial step of the implicit background solver, multiple steps are performed for the explicit overlapping solver, based on convergence criteria. Since the background mesh covers a large percentage of the total domain, this hybrid significantly reduces run-time whilst maintaining simplicity. A comparison of results and run-times for this hybrid solver and the fully explicit solver are shown in table 4.1 in Section 4.2. Implicit and explicit time integration methods have been combined on overset grids by Burton and Eaton [86]. However, in this case, it is the overlapping regions that are given an implicit method to overcome time-step restrictions. The background grids use an explicit time integration scheme. This was aimed at improving the overall convergence of the solution rather than the CPU time required. It is thought that with the current method, by using the unstructured solver for small overlapping meshes only, any effects on the overall solution convergence will be minimal. It was decided that for the background solver, a first order artificial time integration scheme would

be sufficient. The system of equations, with the artificial time derivative included is,

$$A_i \frac{U^{n+1,m+1} - U^{n+1,m}}{\Delta\tau} + A_i I_0 \frac{U^{n+1,m+1} - U^{n}}{\Delta t} = -R(U^{n+1,m+1}). \tag{4.2}$$

Linearising the RHS of equation 4.2 for the artificial time level m+1, using Newton's method yields,

$$[I_m A_i + \frac{\partial R(U^{n+1,m})}{\partial U}](U^{n+1,m+1} - U^{n+1,m}) = -[I_0 A_i \frac{U^{n+1,m} - U^{n}}{\Delta t} + R(U^{n+1,m})], \tag{4.3}$$

where,

$$I_m = \begin{pmatrix} \frac{1}{\Delta\tau} & 0 & 0 \\ 0 & \frac{1}{\Delta\tau} + \frac{1}{\Delta t} & 0 \\ 0 & 0 & \frac{1}{\Delta\tau} + \frac{1}{\Delta t} \end{pmatrix} \tag{4.4}$$

Equation 4.3 can be written in matrix form,

$$(D + L + U)\Delta U^s = RHS \tag{4.5}$$

where D,L and U are the 3x3 block diagonal, lower triangular and upper triangular matrices, respectively. The system is then solved using an approximate lower-upper factorisation (ALU) scheme, as proposed by Pan and Lomax [87],

$$(D + L)D^{-}1(D + U)\Delta U^s = RHS \tag{4.6}$$

4.1.4 Moving Grid Method

When simulating moving bodies, adjustments need to be made to the solver to ensure the body velocity is considered. The method used in the current solver

is the moving grid method [14], chosen for its simplicity. This is where a single coordinate system is used for all grids. At each time-step, the positions of all grid points on a moving mesh must be updated based on the body velocity. An alternative method is to use moving frame of reference [88]. This is where each grid has its own coordinate system (or frame), which moves with the grid, eliminating the requirement for updating positions of grid points. Instead, a moving grid has a velocity relative to a stationary grid and in order to interpolate data between the grids, it must be converted from relative into absolute velocity. Unlike the moving frame of reference method, the moving grid method works with an absolute velocity for all meshes. Instead, the inviscid flux in the governing equations is modified to account for the body velocity in the following way,

$$f^I = \begin{bmatrix} u \\ (u - u_{body})u + p \\ (u - u_{body})v \end{bmatrix}, \quad g^I = \begin{bmatrix} v \\ (v - v_{body})u \\ (v - v_{body})v + p \end{bmatrix} \tag{4.7}$$

where u_{body} and v_{body} are the x and y components of the body velocity, respectively. In addition, the boundary conditions around the body must be amended to give the body walls a velocity equal to the body velocity to result in a relative velocity of zero.

After updating the position of the moving, overlapping grid, the hole-cutting process must be repeated. Any previous hole-cut cells that are now a part of the domain require stored solution data from previous time-steps in order to integrate through time. These cells are given values by interpolating the stored solution from the overlapping mesh using the grid interpolation method (see Section 4.1.2).

4.1.5 Cut-Cell Method for Hole-cutting

The previously used hole-cutting method has many limitations and needs to be replaced by a more sophisticated method. As discussed in Chapter 2, current hole-cutting methods tend to be complex, with many requiring user input and expertise. A novel and alternative approach is considered in the current research. This is to apply the Cartesian cut-cell method to the hole-cutting procedure within the overset approach.

The Cartesian cut-cell method is an approach used to cut a geometry from a Cartesian grid and would ordinarily be an alternative to the overset approach. It slices directly through cells to generate the geometry of a solid object or flow feature and can be used to represent moving bodies by simply repeating the cutting process for each time-step. The method benefits from being automated, simple and fast, but it suffers in representing complex geometries due to sharp edges getting sliced off in the cutting process.

Although the Cartesian cut-cell method is an existing cutting method, it was made for a different purpose. In order to apply it to a hole-cut, some alterations are required, which will be discussed in this section. In combining the overset and cut-cell methods in the proposed manner, it is thought that the disadvantages of the cut-cell method will be eliminated along with problems with hole-cutting for the overset approach. The Cartesian cut-cell method is thought to make a good alternative to conventional hole-cutting methods for the following reasons:

1. It is already a well established approach for cutting a hole in a Cartesian grid.

2. When applying the method to a hole-cut rather than a geometry cut, complex geometries can be avoided. This is because the hole can always be a simple geometry since it is no longer representing a body.

3. The procedure is fully automatic, unlike many current hole-cutting methods.

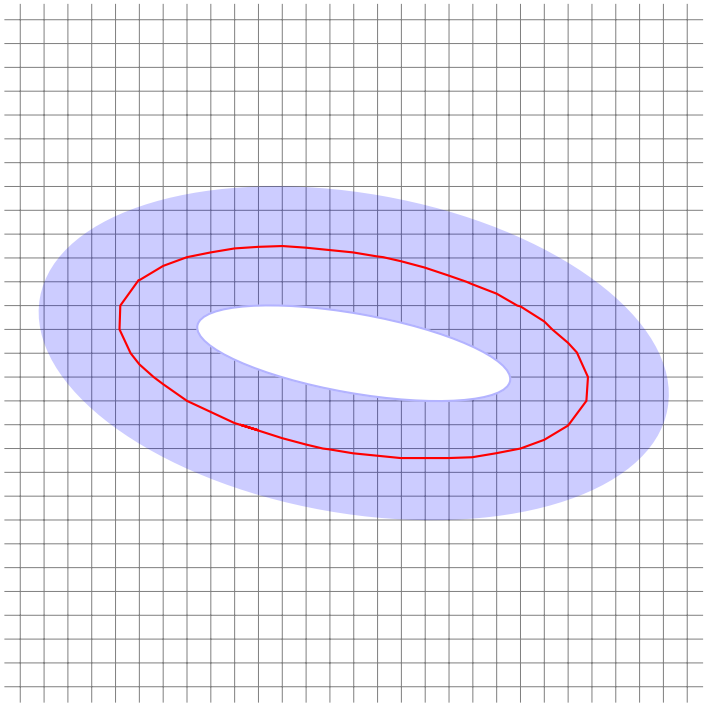
4. Since the hole-cut is made directly through cells rather than around them, the cutting process is simplified.

5. The solution procedure does not get overly complicated by the presence of cut-cells.

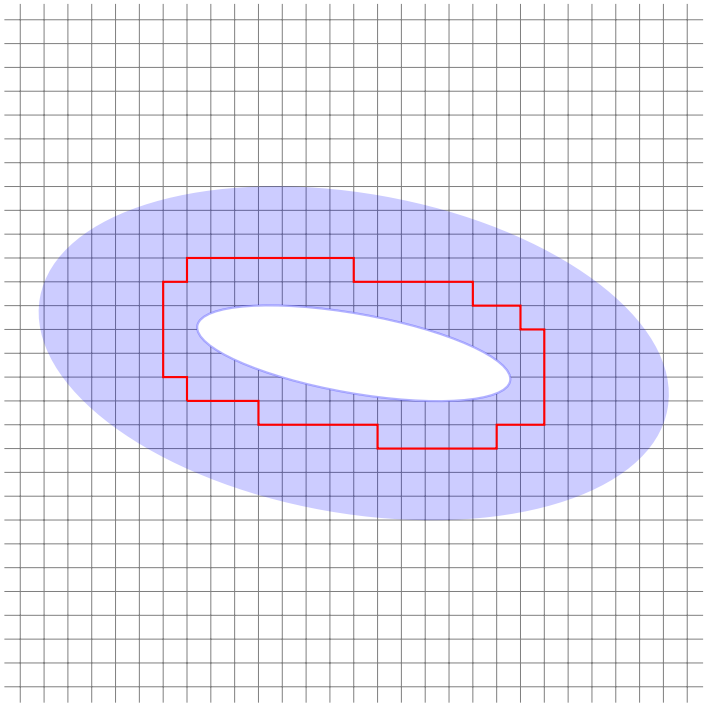
6. Any number of regions or bodies can be cut from a mesh using the cut-cell method, meaning the number of hole-cuts that can be performed is unrestricted, Hence, it is compatible with multiple overlapping grids.

The cut-cell method has not been used for hole-cutting previously within the literature and offers a different hole-cut to existing techniques. The main difference between a cut-cell hole-cut and existing hole-cutting techniques is the way in which the cut is made. In existing methods, the hole is cut around cells, leaving an irregular hole-geometry. In performing a cut-cell hole-cut, the cells will be sliced through, leaving a smooth hole-cut. This is demonstrated by Figure 4.2, which shows a comparison of the proposed cut-cell method for hole-cutting and a typical existing hole-cutting method. In each case, there is an overlapping mesh around an ellipse and the hole is cut from the background grid around this object. One advantage of slicing through cells in the proposed cut-cell method is without having to manoeuvre around cells to perform the cut, the process is simplified. Additionally, it allows for a uniform overlap to be held between grids. Thus, a minimum overlap can be enforced, without the need for an additional optimisation procedure.

This new application of the cut-cell method will work by cutting the hole in the same way it would normally cut an object from a grid in its original purpose.



(a) Proposed cut-cell hole-cut



(b) Typical existing hole-cut

Figure 4.2: A comparison of the proposed and existing hole-cutting techniques. (Blue: overlapping mesh around object, Red: Hole-cut)

However, instead of an object boundary, there is now a hole boundary to be cut. This is demonstrated by Figure 4.3, where the hole-cut (red) can now be seen to slice directly through cells on the background grid. As with the previous hole-cut method, an overlap of no less than two whole cells should been left.

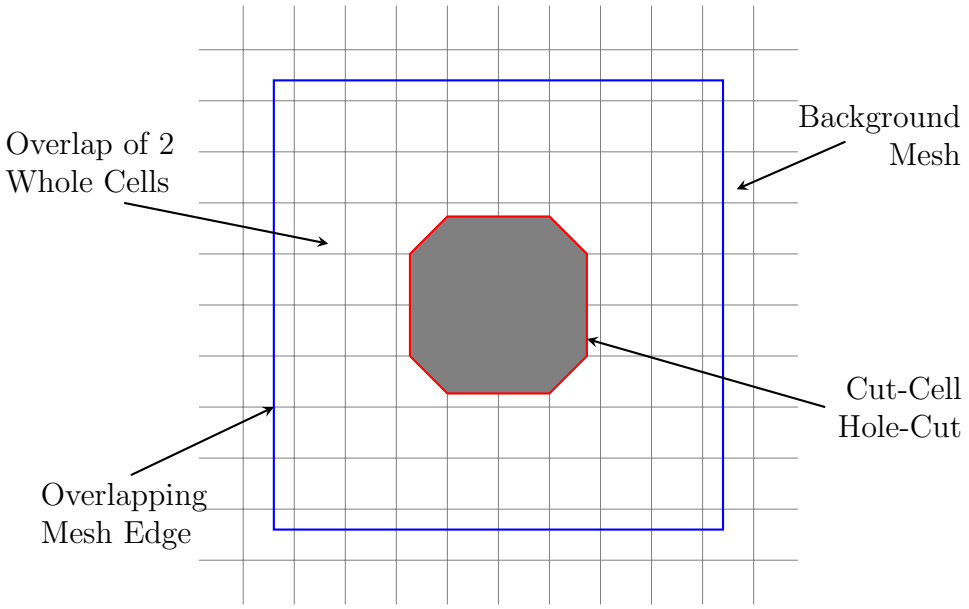


Figure 4.3: Diagram of overset grids, showing a cut-cell hole-cut on background grid.

The hole boundary is defined by a set of poly-lines, given in an anti-clockwise direction. The sets of poly-lines are currently input by the user initially, but they could be calculated by a distance function from the overlapping mesh edge or solid body edge. For a moving body problem these poly-lines are automatically updated using the body velocity. The hole is fixed to the overlapping mesh and is re-cut into the background mesh as it moves. To perform the cut, intersection points need to be found on the grid for each line segment contained within the set of poly-lines defining the hole boundary. Once all intersections have been determined, they are connected to form an enclosed hole. These connections of the intersections form the cuts to be made to the mesh, and the cells in which

they pass through are labelled as cut-cells. However, no physical cuts are made to the grid. Instead, a list of cut-cells and cut-cell data is stored for use in the solution procedure. This cell data includes new cell side lengths, area and the centroid location. Additionally, new cell gradients are calculated to allow for 2nd order accuracy. Cells outside of the hole-cut are labelled as fluid cells and are treated in the usual manner. Cells within the hole-boundary are labelled as solid cells and treated in the same way they would be in the original method, in order to exclude them from the solution calculations. This whole process is given in more detail in Section 2.3.1, since it is the same as in the original cut-cell purpose.

It is within the treatment of cut-cells that alterations need to be made to the original cut-cell method. The cut-cells are now bounding a hole rather than a solid object. In the original method, solid boundary conditions are enforced and there is no flux computed at the oblique edge. In the overset method, there needs to be flow through the hole boundary to allow for data to be transferred between grids. This is achieved by mirroring the cut-cells over the hole-boundary to form ghost cells, as shown by Figure 4.4.

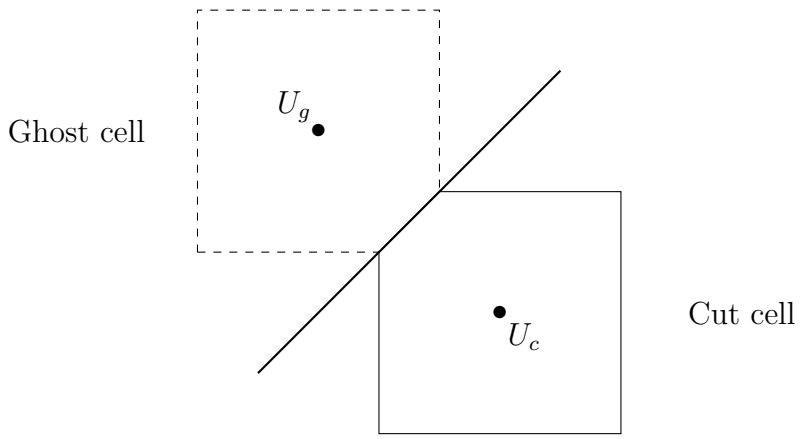


Figure 4.4: Illustration of ghost cell formation by reflecting cut-cells over the hole-boundary

U_c and U_g are the stored values at the centres of the cut cell and ghost cell,

respectively. Ghost cell data is computed by linear interpolation from the overlapping grid using the approach described in Section 4.1.2. Since data is known for these ghost cells, no boundary conditions are used for the hole-boundary. The flux through the oblique edges cannot be obtained in the usual way for Cartesian grids, since they do not conform to the x and y axis. Instead the approach used for the unstructured grids solver is applied, where flux can be calculated in a general direction. Details of this can be found in Section 3.1. This flux is then simply totalled with the flux through the other sides for that cell.

In some instances, the cutting procedure can result in very small cells, which can cause time-step stability issues. A solution to this is to use a cell merging technique, which was developed by Yang et al. [36] for compressible flows. The technique was later applied to incompressible flow problems and with extensive validations carried out [24], [89]. Cell merging is where the small cells whose area is less than a chosen tolerance A_{min} are merged with a suitable neighbouring cell. The choice of A_{min} is based upon a consideration of both the time-step and the geometric resolution. For this solver, a low tolerance of $0.001\Delta x\Delta y$ is used.

For moving body problems, once the hole has been re-cut there will be cells that were within the hole region in the previous time-step but are fluid or cut cells in the current time-step. This means that no solution data is available for the previous step. To obtain data for the time-integration process, the solution for the previous time-step on the overlapping mesh is interpolated to these cells. This is performed in the same way as the grid interpolation discussed in Section 4.1.2.

A known issue with the original cut-cell method is that it cannot accurately represent complex geometries. This is because and sharp edges in a geometry will be represented by two line segments intersecting a cell. The Cartesian cut-cell method can only perform one slice per cell, meaning in this case, an approximation

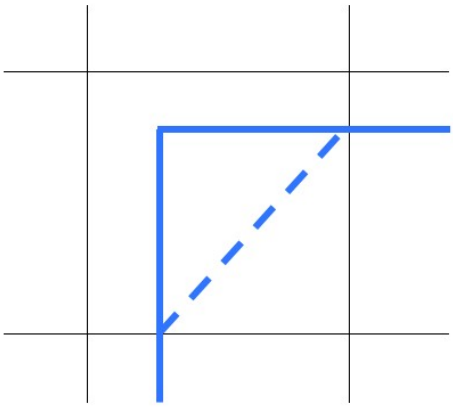


Figure 4.5: Example of a cut-cell approximation (solid line: line segments, dashed line: cut made)

has to be made as shown by Figure 4.5. For this reason, sharp edges should be avoided as they cannot accurately be represented and may cause inaccuracies in the solution.

4.1.6 Code Structure

For the solver that is fully explicit in artificial time, the code structure is fairly simple and is illustrated by the flowchart in Figure 4.6. For every real time-step, multiple artificial time-steps are performed. For each artificial time-step, both the overlapping mesh and background mesh are solved. Once convergence criteria has been met for both meshes, the solver will move on to the next real time-step and repeat the process.

For the solver that is a hybrid of implicit and explicit artificial time-steps, a change has to be made to the code structures, as shown in the flow chart in Figure 4.7. Now, for every implicit artificial time-step on the background mesh, multiple explicit time-steps are performed on the overlapping mesh. This is because the

implicit time integration is much less restrictive on the artificial time-step and converges faster than the explicit time integration. Allowing multiple explicit time-steps for every implicit ensures the overlapping grid converges without having to perform many unnecessary iterations on the background grid.

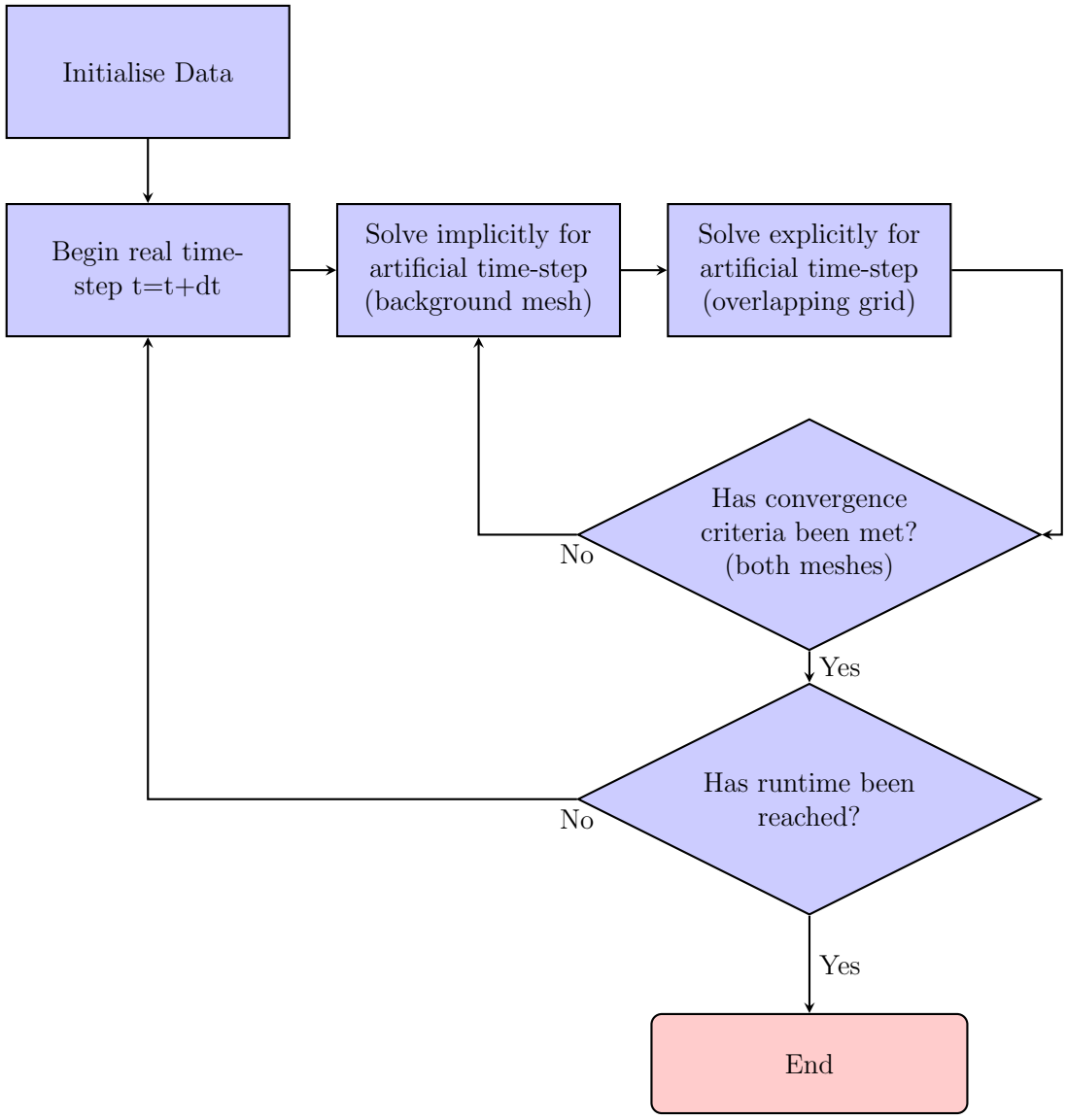


Figure 4.6: Flow chart of code structure for fully explicit solver.

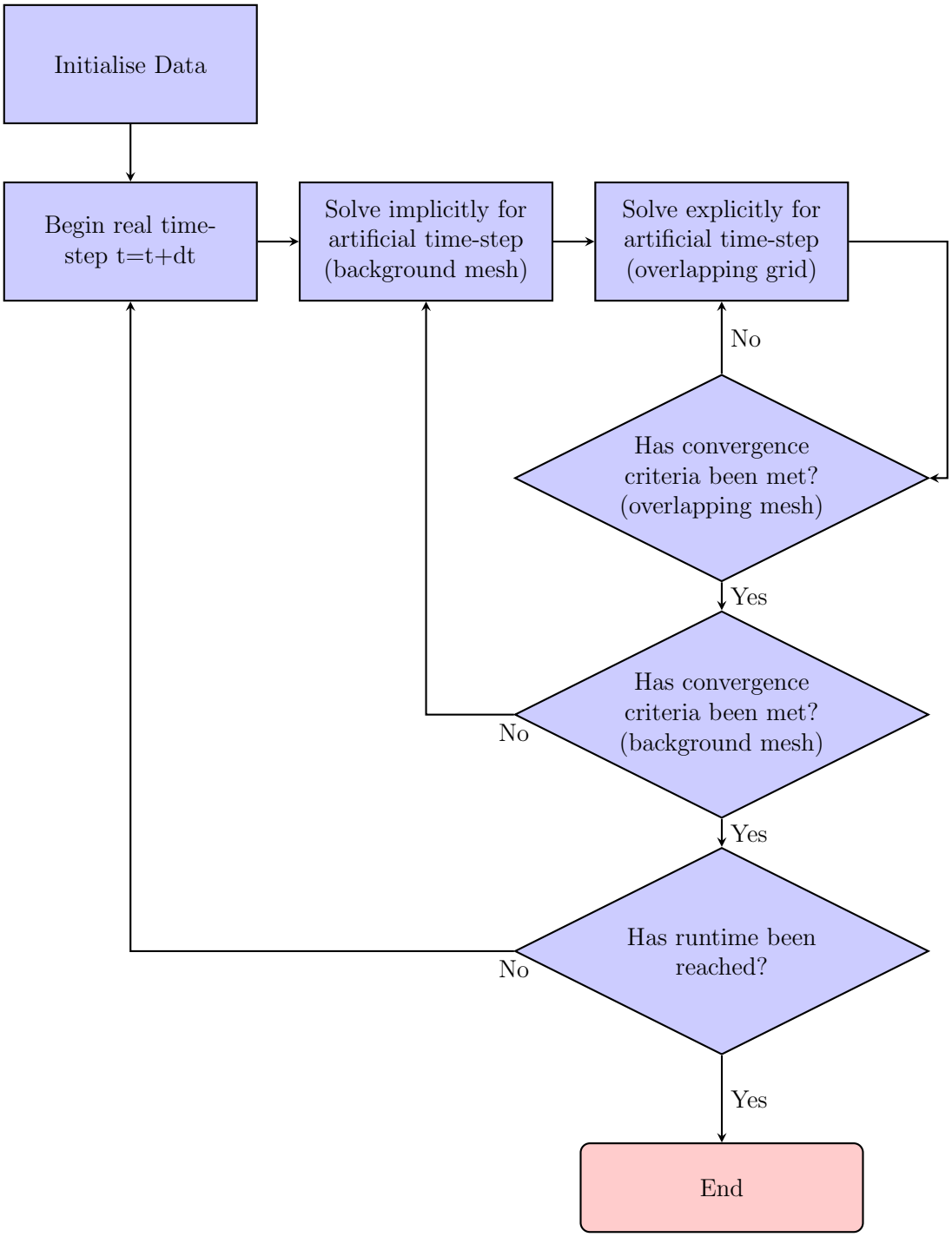


Figure 4.7: Flow chart of code structure for implicit/explicit hybrid solver.

4.2 Overset Grids Validation Test Results

4.2.1 Introduction

In this section, benchmark tests are performed to validate the overset mesh solver. The first tests are the flow past a stationary and oscillating cylinder problems. The stationary case was chosen for its simplicity and to make a direct comparison to the unstructured solver, developed in Chapter 3. The oscillating case was chosen to test the ability of the overset solver to deal with a moving body, which is its main purpose. These tests are performed at several stages of the development process. Firstly, with fully explicit artificial time integration and the simple hole-cutting method discussed in Section 4.1.2. Then, with a hybrid of time integration methods, as discussed in Section 4.1.3. And finally, with the novel cut-cell hole-cutting method as described in section 4.1.5.

4.2.2 Flow Past a Stationary Cylinder

This test, already performed with the unstructured solver, has been repeated for the new overset solver. Details of the problem setup are provided in Section 3.2.5. A real time-step of 0.1 and a β value of 1 is used to perform these tests for Reynolds numbers of 185, 300, 500 & 1000.

Simple Hole-Cutting Method

The first test is from the earlier stages of the overset development, which is fully explicit in artificial time and utilises the basic hole-cut method. Upon running this simulation for Re=185, it was discovered that the code was still far too slow, taking approximately 42 hours of CPU time to run to t=200 (PC Intel(R) Xeon(R) Quad Core CPU X5450 @ 3.00GHz, 500GB hard disk, 4GB RAM). This

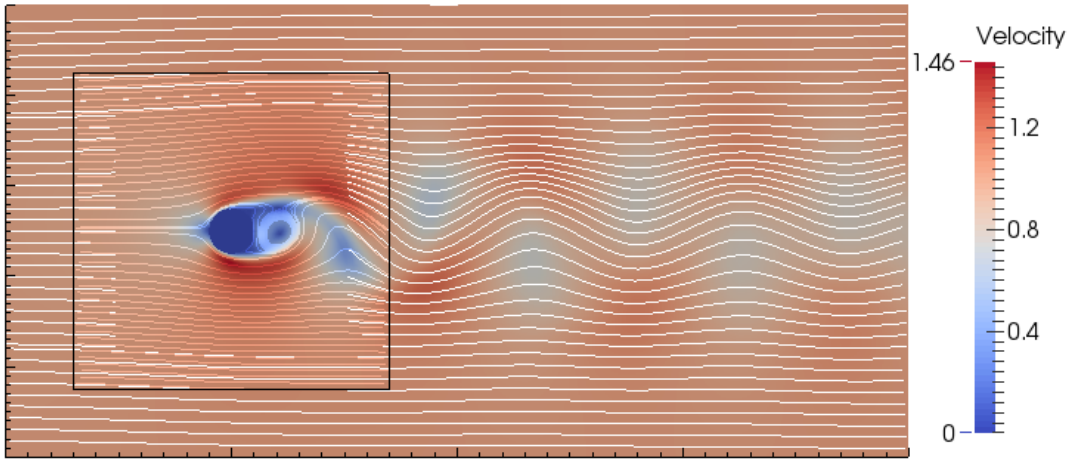
problem was first discussed in Section 3.2.6, where it was concluded that once the solver was only being used for smaller overlapping grids, it may no longer be an issue. However, these results show that not to be the case. The solution found for this problem was to introduce a hybrid of artificial time integration methods. Details of this are in Section 4.1.3. Table 4.1 gives a comparison of the results obtained by each method along with the corresponding CPU times. It also contains the previously found results for the unstructured single mesh solver, which was not timed. Here it can be seen that there is a vast improvement in efficiency with the new hybrid, with the CPU time dropping to just 4.6 hours for the same test run on the same machine. There is a slight difference in the average drag coefficient and the Strouhal number found by each overset method, but the difference is small enough that the improvement in computational time far outweighs this. There does appear to be a larger difference between the drag coefficient values for the single unstructured solver and the overset solvers.

	Re=185		
	$\bar{C}_D$	St	CPU time (hrs)
Fully Explicit	1.31	0.189	42.0
Implicit/Explicit Hybrid	1.30	0.187	4.6
Single Unstructured	1.38	0.189	-

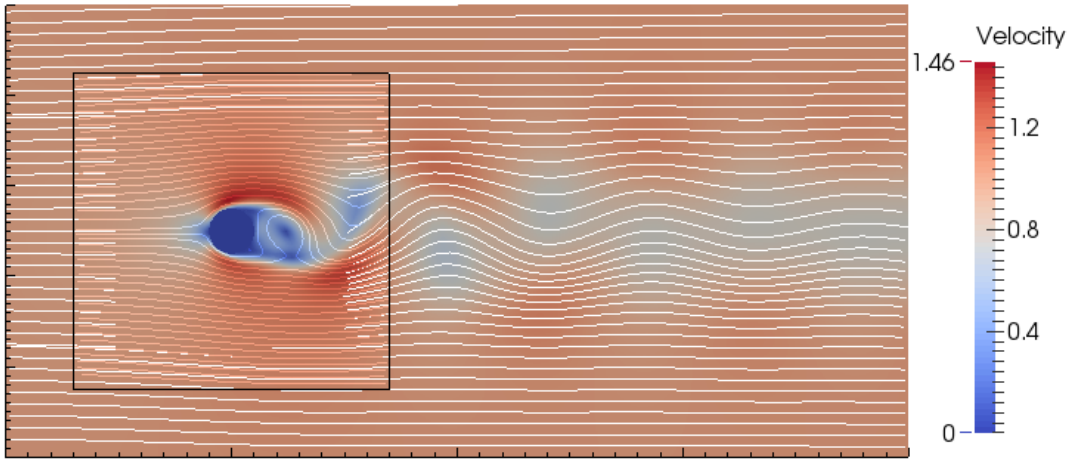
Table 4.1: A comparison of CPU time and results for the fully explicit solver and the implicit/explicit hybrid solver.

Figure 4.8 shows the solution provided by each overset solver at t=200 for comparison. The black box on each represents the outline of the overlapping mesh. The streamlines on the overlapping mesh are coloured to match the velocity, and the background grid streamlines are left in a solid colour. This is so they can be compared in the overlapping region. In both cases they agree well, suggesting that the solution is transferred well across the grids. It can be observed

that the tips of the background streamlines are not aligned with the streamlines for the overlapping grid. However, these tips lie within the hole-cut region, where velocity values are set to zero. For this reason they should be ignored as they are not a part of the solution.



(a) Fully explicit solver



(b) Implicit/explicit hybrid solver

Figure 4.8: Velocity streamlines of solution at t=200 for each time integration method using simple hole-cutting (Re=185).

Figure 4.9 shows a plot of the drag and lift coefficients over time for each solver, including the single mesh unstructured solver. Here it can be seen that vortex shedding occurs sooner for the new hybrid solver, than the two previous solvers. A plot of the drag and lift coefficients over time from published work by Guilmineau

& Queutey [1] is given in Figure 4.10 for visual comparison with the present results. Here it can be seen that vortex shedding appears to begin close to the time found using the implicit/explicit hybrid solver.

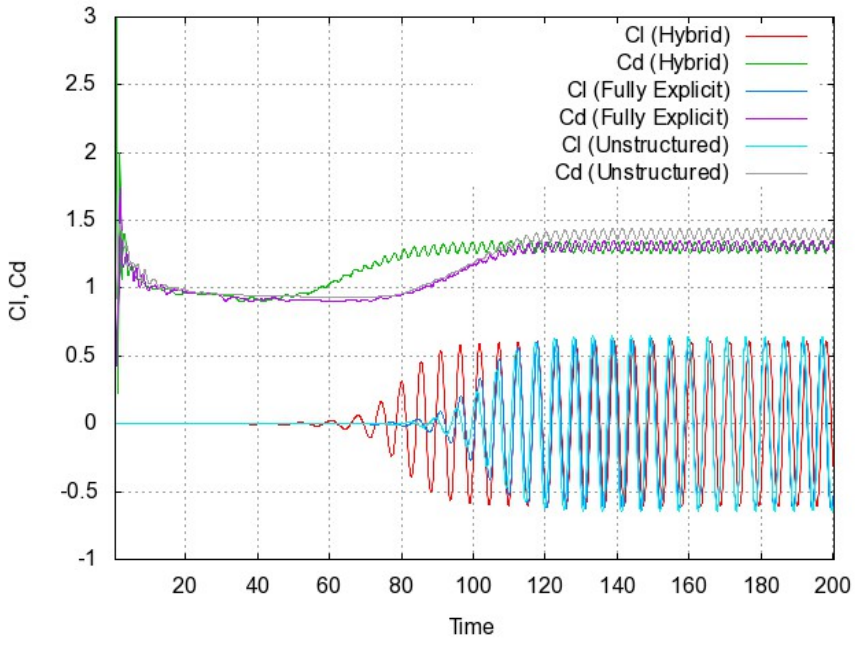


Figure 4.9: Comparison of the drag and lift coefficients over time for the implicit/explicit hybrid, fully explicit & single unstructured solvers (Re=185).

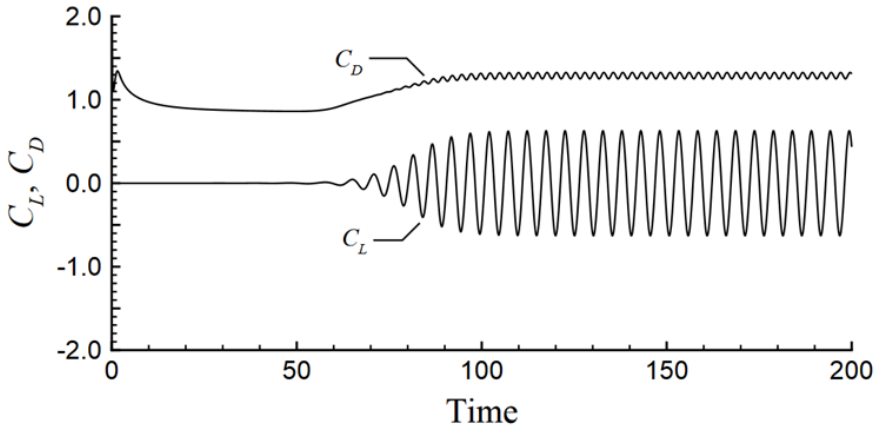
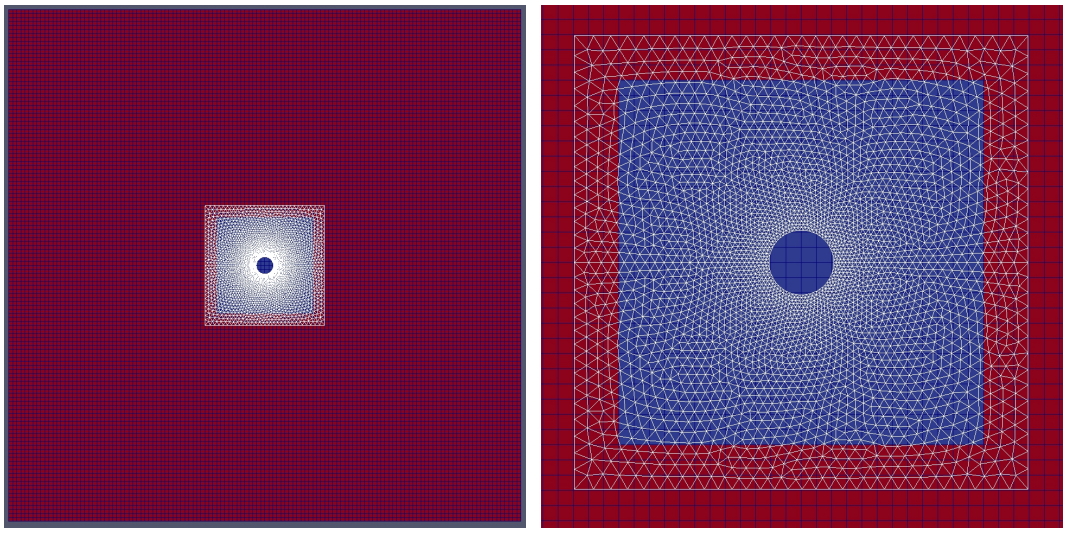


Figure 4.10: Plot of drag and lift coefficients over time from published work by Guilmineau & Queutey [1] (Re=185).

For this comparison, the meshes used were chosen through mesh convergence tests using the faster hybrid solver. The mesh converged solution for Re=185 was



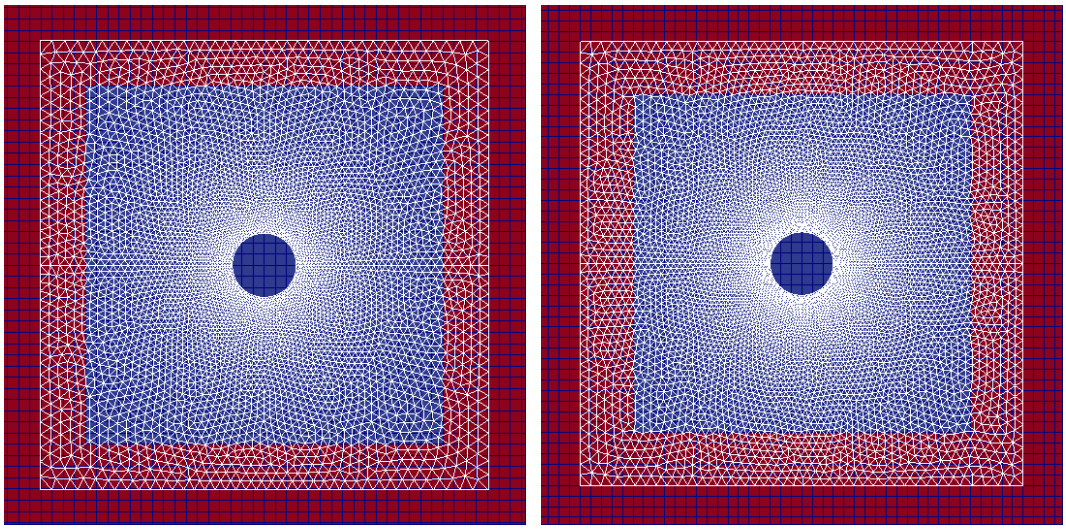
(a) Full Overlapping Meshes (b) Magnified View of Meshes

Figure 4.11: The overlapping meshes used to obtain a mesh independent solution for Re=185.

found on overlapping meshes with a combined total of 22,422 cells as shown by Figure 4.11. Only a small section of the background mesh (red), which covers the 30×30 domain is shown here so that the overlapping mesh (wire-frame) can be clearly seen. The overlapping mesh is of non-uniform distribution, with a total of 6,038 cells covering a domain of 7×7, central to the whole domain. This mesh has a smallest cell side length of 0.05 around the cylinder, which is in the centre of the mesh. The background mesh is uniformly distributed with 128×128 cells with a hole cut (blue) from the region under below the overlapping grid. For this simple hole-cutting technique an overlap of at least two whole cells was held between the grids. This value was found through experimentation to be the minimum overlap needed to maintain accuracy. A larger overlap would mean a less efficient solver. This is because the overlapping area gets solved twice; once with each solver. The convergence tests were based on the average drag coefficient ($\bar{C}_D$) and Strouhal number (St).

Mesh convergence tests were also performed for Reynolds numbers of 300, 500

and 1000. For Re=300, the current overlapping meshes were found to provide an independent solution. However, for Re=500 & 1000 the meshes needed to be refined. An independent solution was found for Re=500 on overset grids with a combined total of 39,350 cells. The overlapping mesh has a smallest cell side length of 0.04 and a total of 9,766 cells. The background grid consists of 172 $\times$ 172 cells. For Re=1000, the meshes needed to be further refined to obtain a mesh independent solution. The refined overset grids have a total of cells, 45,134 of which make up the overlapping mesh, which has a smallest cell side length of 0.035. The background grid consists of 180 $\times$ 180 cells. Figure 4.12 shows magnified views of these overlapping grids used for Re=500 & 1000. It should be noted that since there is only a small increase in the total number of cells between these meshes, a further test was performed on a finer grid for Re=500, after a mesh independent solution was thought to have been found, to ensure this was the case.



(a) Re=500 (b) Re=1000

Figure 4.12: The overlapping meshes used to obtain a mesh independent solution for given Reynolds numbers

$Min\Delta S$	Non-Uniform Factor	No.Cells (overlapping)	No.Cells (background)	No.Cells Total	Re=185		Re=300		Re=500		Re=1000	
					$\bar{C}_D$	St	$\bar{C}_D$	St	$\bar{C}_D$	St	$\bar{C}_D$	St
0.07	1.0	3,090	102^2	13,498	1.36	0.187	-	-	-	-	-	-
0.06	1.0	4,322	112^2	16,866	1.31	0.187	1.37	0.202	1.30	0.220	-	-
0.05	1.0	6,038	128^2	22,422	1.30	0.187	1.35	0.202	1.40	0.218	1.05	0.253
0.04	1.0	9,766	172^2	39,350	1.30	0.187	1.35	0.202	1.45	0.212	1.51	0.238
0.05	0.6	10,182	206^2	52,618	1.30	0.187	-	-	-	-	-	-
0.35	1.0	12,734	180^2	45,134	-	-	-	-	1.45	0.212	1.51	0.230
0.3	1.0	17,290	206^2	59,726	-	-	-	-	1.45	0.212	1.51	0.230

Table 4.2: Mesh convergence tests using average drag coefficient and Strouhal number.

Details of all of the mesh convergence tests, as well as further mesh data are provided by table 4.2. The tests were conducted using a methodical approach; the minimum cell side length (around the cylinder) was steadily reduced, whilst keeping the non-uniform factor constant. In doing this, the number of cells contained in the overlapping mesh was gradually increased, resulting in larger cell side lengths at the boundaries. The cell size of the uniform, Cartesian background mesh was then reduced so that the cell side lengths roughly matched those of the overlapping boundary (interface). Once it appeared that a mesh converged solution had been obtained for Re=185, the non-uniform factor was reduced to ensure this was not having any impact. The initial value for this factor was chosen to give adequate smoothness to the stretching of the mesh, whilst maintaining cell side lengths at the boundaries, which result in a background mesh of similar quality.

Plots of the drag and lift coefficients over time obtained for Re=300, 500 & 1000 are given in Figures 4.13, 4.14 and 4.15, respectively. These were the mesh independent solutions found. A comparison has been made to published numerical results for all of the Reynolds numbers, which are given in table 4.3.

Careful consideration was taken in deciding upon the overlapping mesh size. In theory, a smaller mesh should increase efficiency and overall accuracy, since it reduces the size of the overlapping region and thus, decreases the number cells to interpolate data between. However, this is not always the case. Another factor in determining the efficiency of the solver is the background grid cell size. This needs to be fine enough to produce accurate results, but if it is needlessly fine, efficiency is lost. Since the background grid cell size is determined by the side lengths at the edges of the overlapping mesh, this mesh can play a large role in the overall efficiency. If the cell side lengths are too small here, the background grid will be too fine. These boundary side lengths are determined by three factors, the

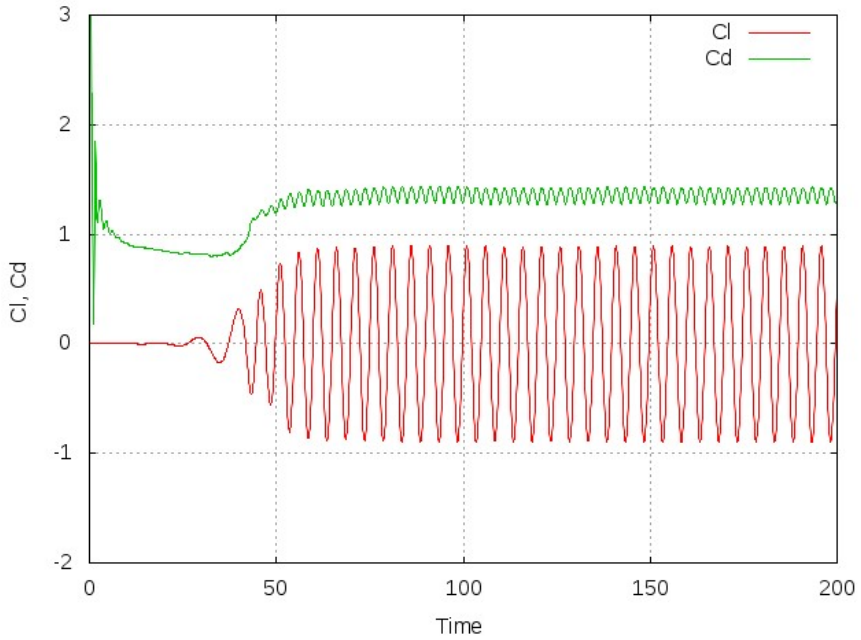


Figure 4.13: Plot of drag and lift coefficients over time using simple hole-cut for Re=300.

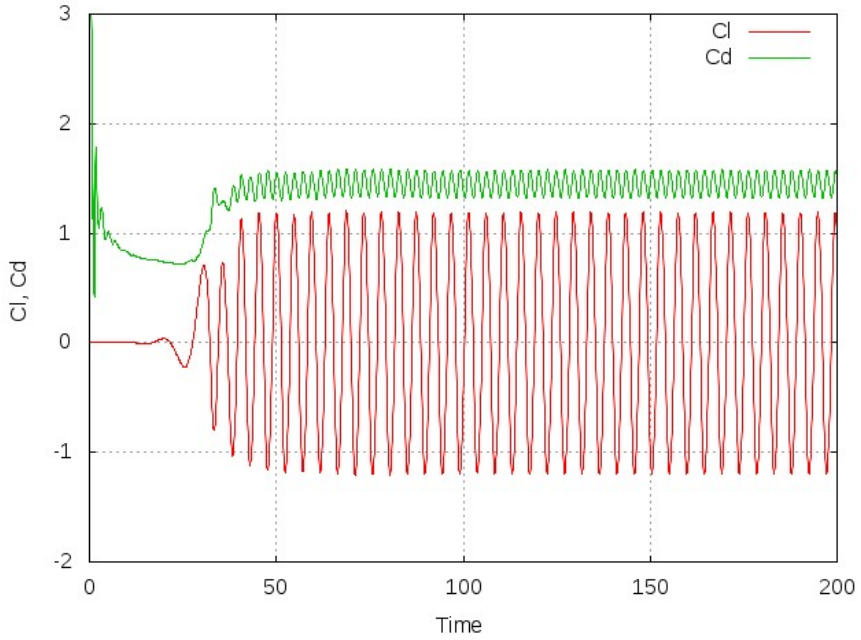


Figure 4.14: Plot of drag and lift coefficients over time using simple hole-cut for Re=500.

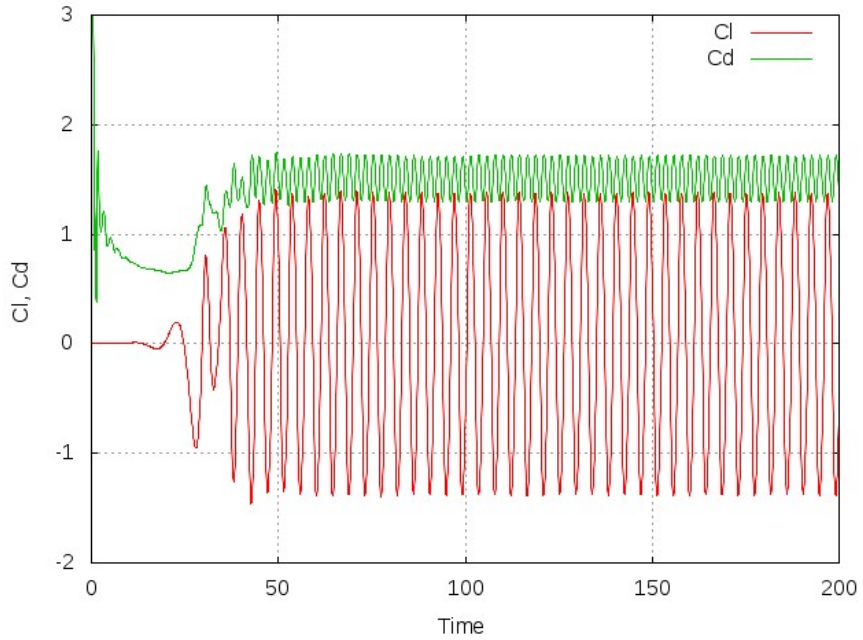


Figure 4.15: Plot of drag and lift coefficients over time using simple hole-cut for Re=1000.

	Re=185		Re=300		Re=500		Re=1000	
	$\bar{C}_D$	St	$\bar{C}_D$	St	$\bar{C}_D$	St	$\bar{C}_D$	St
Chung [80]	1.375	0.184	-	-	-	-	-	-
Guilmineau & Queutey [1]	1.287	0.195	-	-	-	-	-	-
Rajani et al. [90]	-	-	1.37	0.215	-	-	-	-
Mittal & [91] Balachandar	-	-	1.38	0.213	-	-	-	-
Qian & Vezza [92]	-	-	-	-	-	-	1.52	0.240
Lu & Dalton [83]	1.31	0.195	-	-	1.22	0.222	1.21	0.224
Present	1.30	0.187	1.35	0.202	1.45	0.213	1.51	0.230

Table 4.3: Drag coefficients and Strouhal numbers for present solver compared with published results

minimum cell side length, the stretching factor and the mesh size. The minimum cell side length needs to be relatively small to capture the solution around the cylinder. If the stretching factor is too large, the mesh loses quality and as a

result, accuracy can be lost in the solution. The factor that has more flexibility is the size of the overlapping mesh. A larger mesh allows for a reasonable cell side length to be generated at the boundary using a smaller stretching factor and maintaining a small minimum side length. Initially, for thee current test, an overlapping mesh size of 3×3 was chosen based on the initial theory of smaller being more efficient. However, when refining the mesh to obtain a mesh converged solution, the problems became apparent. This was replaced with an overlapping mesh size of 7×7, which was chosen through experimentation in the mesh generation process. An interesting study would be to perform some strict experiments on the effects of the overlapping grid size on the overall solution. The aim of this would be to construct a formula for the the most efficient mesh size. However, this would be very time consuming and was not feasible within the current work due to time restrictions.

Cut-Cell Hole-Cutting Method.

After introducing the new hole-cutting method, the stationary cylinder case was again repeated. This was performed on the same meshes as the earlier tests for the same Reynolds numbers. However, with this new hole-cutting method, a set of poly-lines needed to be input to define the boundary of the hole to be cut. In this case, a square hole was chosen for simplicity and for comparison of results with the old method. Since it had already been observed that an overlap of at least two cells was required, the hole boundary was positioned to give an overlap of 0.7 which is approximately 3 cell side lengths. This ensured that the overlap was never less than 2 whole cells.

Figure 4.16 shows the velocity streamlines of the solution at t=200 and Figure 4.17 shows the drag and lift coefficients over time, both for Re=185. In this case, the average drag coefficient and the Strouhal number were found to be 1.31 and

0.189, respectively. These are the same values obtained with the old hole-cutting method with the fully explicit time integration method.

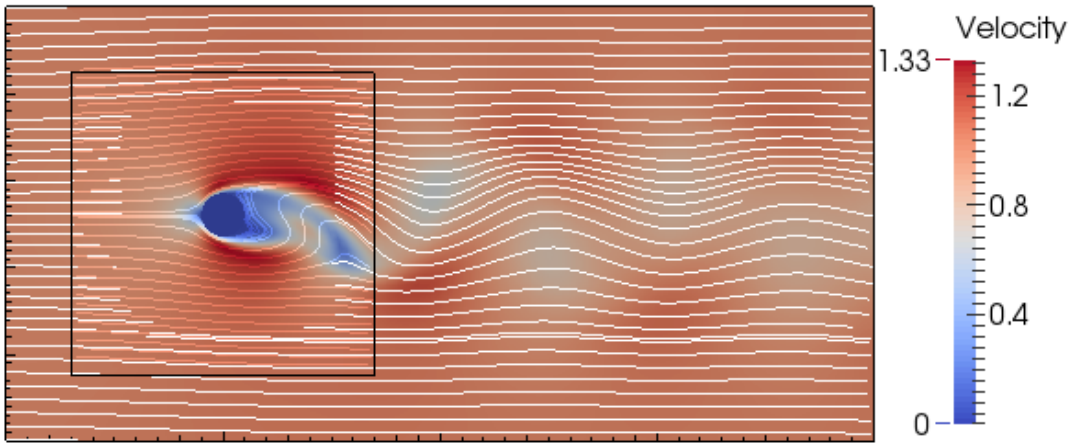


Figure 4.16: Velocity streamlines of the flow past a stationary cylinder at t=200 using cut-cell hole-cutting (Re=185).

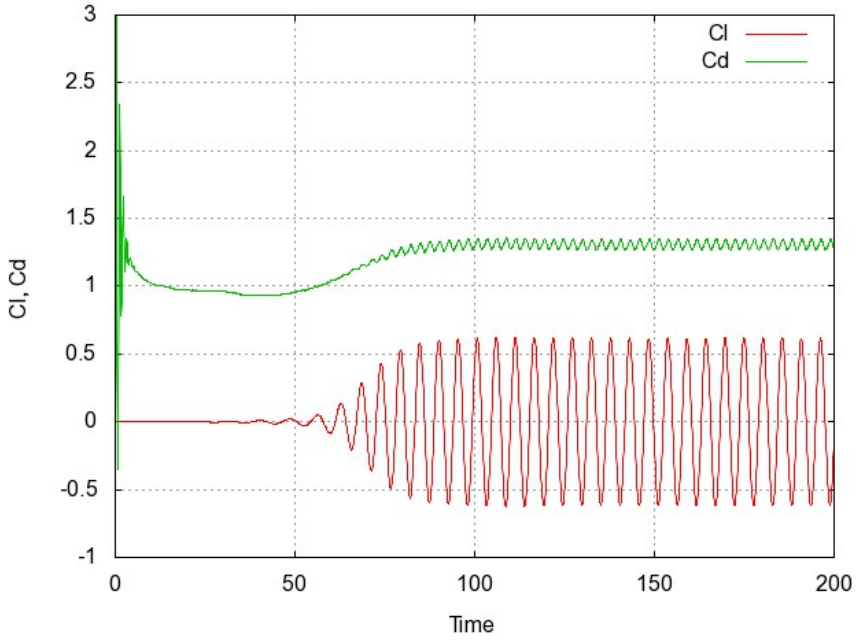


Figure 4.17: Plot of the drag and lift coefficients over time for the new hole-cut method (Re=185).

Figures 4.18, 4.19 and 4.20 show plots of the drag and lift coefficients over time for Re = 300, 500 & 1000, respectively. The average drag coefficient and

Strouhal number for all Reynolds numbers are given in table 4.4, along with values previously obtained by the simple hole-cut for comparison. Here it can be seen that there is a slight difference in values for the two different hole-cutting methods for the lower Reynolds numbers, but for the larger ones, the values are identical.

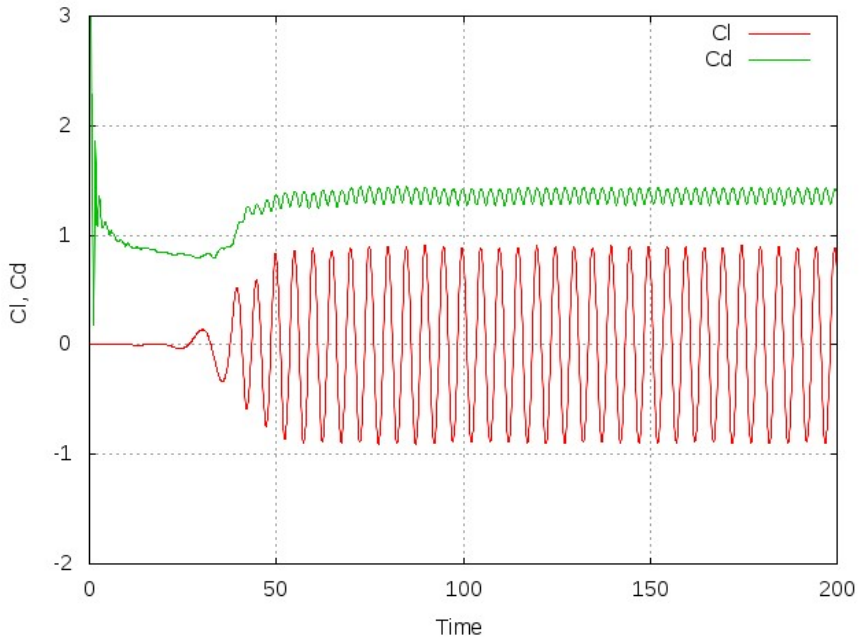


Figure 4.18: Plot of the drag and lift coefficients over time for the cut-cell hole-cut method (Re=300).

	Re=185		Re=300		Re=500		Re=1000	
	$\bar{C}_D$	St	$\bar{C}_D$	St	$\bar{C}_D$	St	$\bar{C}_D$	St
Simple Hole-Cut	1.30	0.187	1.35	0.202	1.45	0.213	1.51	0.230
Cut-Cell Hole-Cut	1.31	0.189	1.36	0.200	1.45	0.213	1.51	0.230

Table 4.4: Comparison of present cut-cell hole-cutting results with previous simple hole-cutting results for all Reynolds numbers.

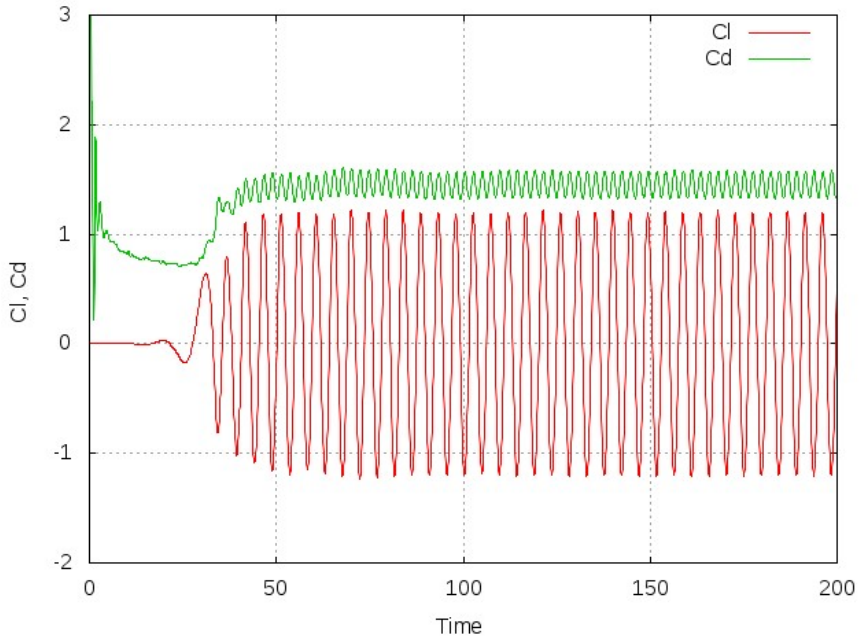


Figure 4.19: Plot of the drag and lift coefficients over time for the cut-cell hole-cut method (Re=500).

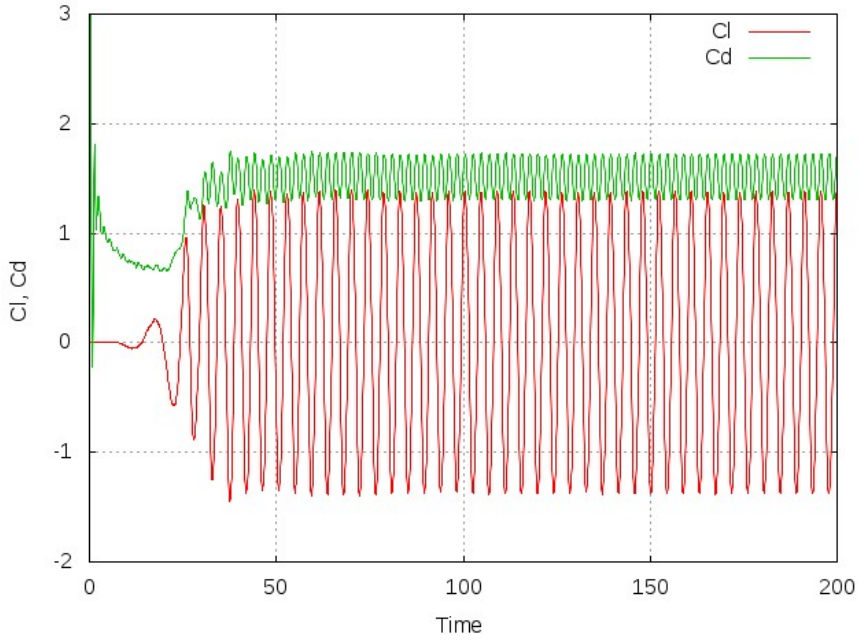


Figure 4.20: Plot of the drag and lift coefficients over time for the cut-cell hole-cut method (Re=1000).

4.2.3 Flow Past an Oscillating Cylinder

This test case is set up in the same way as the stationary case. The only difference is that now the cylinder oscillates perpendicular to the flow direction and hence, the hole must be re-cut at each real time-step. The new centre position of the cylinder, y(t), is updated using the following equation,

$$y(t) = Asin(2\pi f_e t) \tag{4.8}$$

where A and f_e are the amplitude and frequency of the oscillations, respectively. The frequency is calculated using,

$$F = \frac{f_e}{f_0} \tag{4.9}$$

where f_0 is the natural shedding frequency from the stationary case of the same Reynolds number. $A = 0.2$ and $f_e = 0.8$, 1.0 & 1.2 along with a Reynolds number of 185. These values were chosen for comparison with published results. As with the stationary case, a real time-step of 0.1 is used, along with β=1. This oscillating case is tested with both the simple and cut-cell hole-cutting methods and uses the same meshes as the stationary case for the same Reynolds number.

Simple Hole-Cutting Method

In this test using the simple hole-cutting method, again an overlap of at least two cells was chosen and re-cut as required. In this case $f_0 = 0.187$. Figure 4.21 shows the velocity streamlines of the solution obtained at t = 300 for a frequency of 0.8. The outline of the overlapping mesh is given by a black line so that the interface between the grids is clear. Also, the velocity streamlines for the overlapping mesh are coloured to match the velocity, where as the background grid

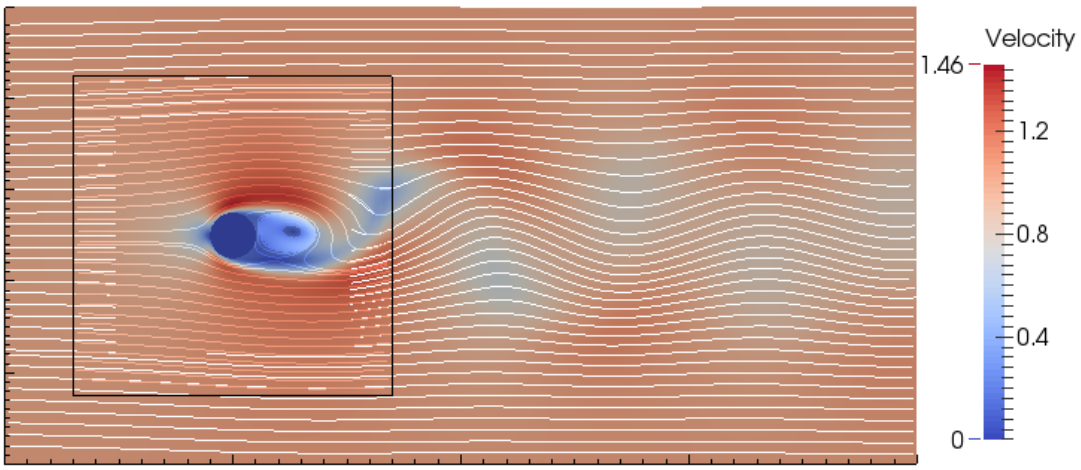
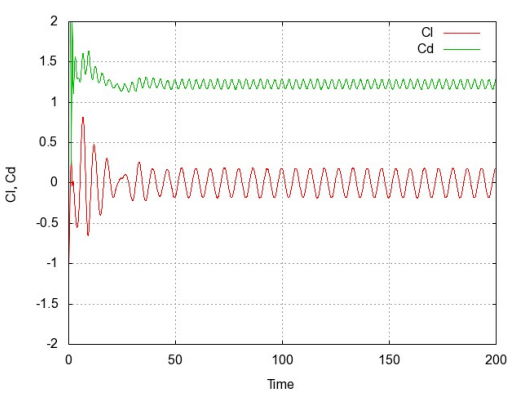


Figure 4.21: Velocity streamlines of the flow past an oscillating cylinder at t=300 using $f_e = 0.8$, with simple hole-cutting (Re=185)

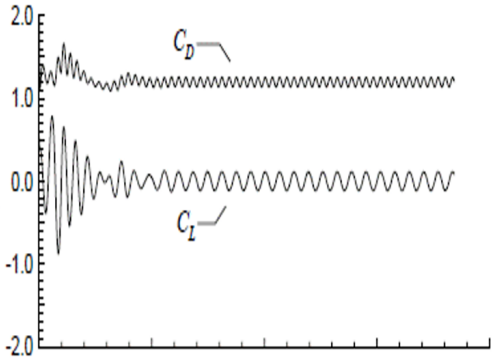
streamlines are a solid colour to easily differentiate between them. The velocity streamlines pass over this interface smoothly, indicating that grid communication is maintained and of a reasonable quality. It should be noted that the tips of the background streamlines are within the hole-cut, where values are set to zero. For this reason, they should be disregarded. Figure 4.22 shows plots of the drag and lift coefficients over time for each of the frequencies. Plots provided by Guilmineau & Queutey [1] have also been included for comparison. Here it can be seen that all three frequencies are producing oscillatory patterns in good agreement with the published results.

Cut-Cell Hole-Cutting Method

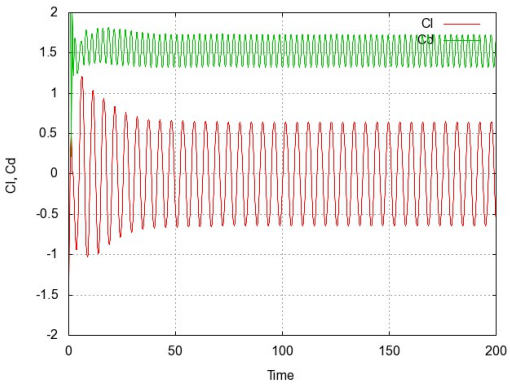
For the test with the cut-cell hole cutting method, the initial hole boundary was set to the same as for the stationary case. The poly-lines were updated automatically at each time-step, using equation 4.8 before the new hole-cut was performed. In this case $f_0 = 0.189$. As discussed in Sections 2.3.1 and 4.1.5, when using the cut-cell method, sharp edges can cause inaccuracies in the solution. This problem was observed within this test case. Figure 4.23 shows the solution to the test



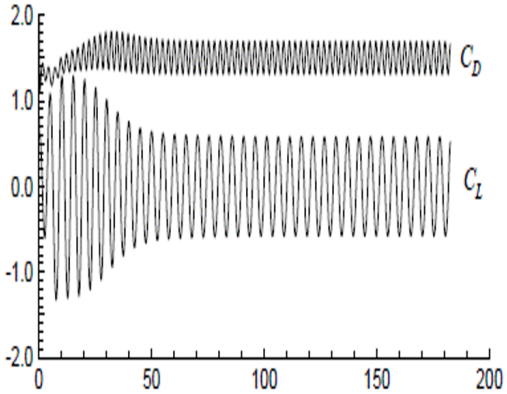
(a) Present, $f_e = 0.8$



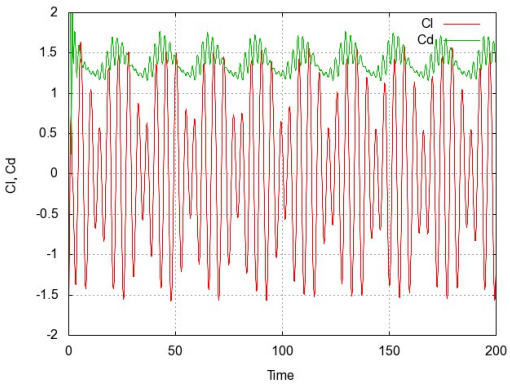
(b) Guilmineau & Queutey, $f_e = 0.8$



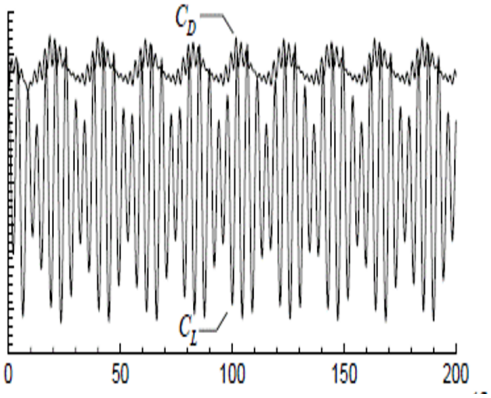
(c) Present, $f_e = 1.0$



(d) Guilmineau & Queutey, $f_e = 1.0$



(e) Present, $f_e = 1.2$



(f) Guilmineau & Queutey, $f_e = 1.2$

Figure 4.22: Plots of drag and lift coefficients over time for present results compared with published using simple hole-cutting (Re=185).

case after just three time-steps using $f_e = 0.8$, where a large error can be seen, stemming from the upper corners of the hole-cut region. In these time-steps, the hole-cut is moving. When the cut moves in a downward direction, an error

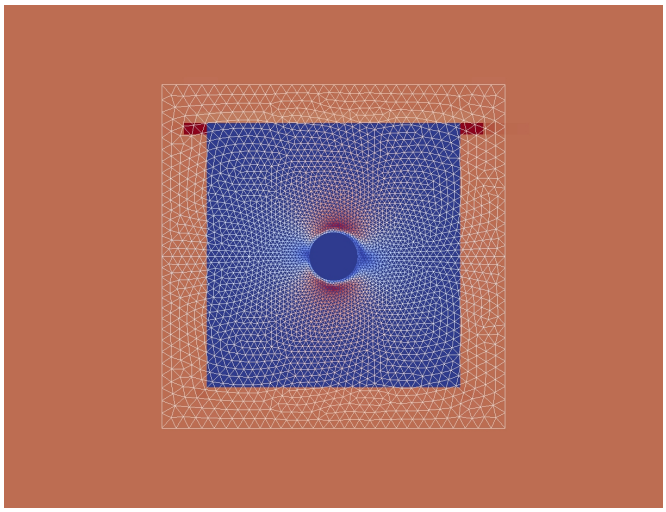


Figure 4.23: Error formation due to moving hole-cut boundary.

forms at the lower corners of the hole-cut region. This error was not present in the solution from the simple hole-cut. The simple reason for this error is that when the hole-cut is performed, the corners of the square originally defined by poly-lines get sliced off. Since the hole-boundary is now treated as a moving boundary, the position of the boundary calculated by the body velocity does not match the position of the cuts made at these corners. This results in an error in the solution, which grows with every time-step. Figure 4.24 illustrates the issue, where the solid lines show where the boundaries should be according to the body velocity and the dashed lines represent the hole-cuts made. The error created here is clear; since there is only a y-directional body velocity present,the cut lines should be parallel to one another. However, this error can be avoided by using rounded corners to define the hole-boundary initially. Figure 4.25 gives the velocity streamlines for $f_e = 0.8$ at t=300, with rounded corners applied. The error is no longer present in the solution. As with the old hole-cutting method, the solution is transferred across the interface well. Figure 4.26 provides plots for the drag and lift coefficients over time, again in comparison to the published results [1], which again are in good agreement.

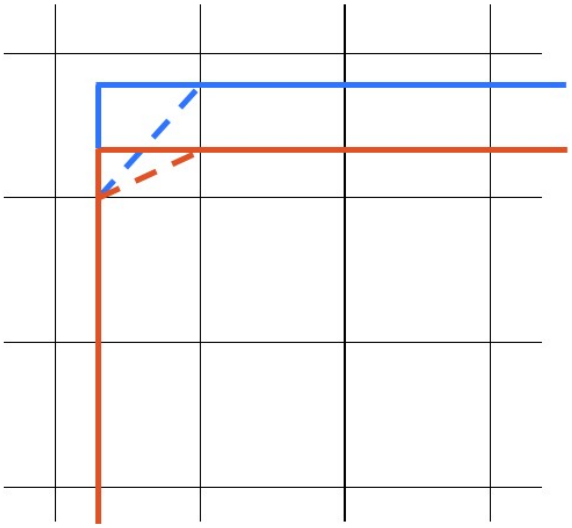


Figure 4.24: Two consecutive hole-cuts (orange & blue) performed due to a moving overlapping grid. Solid line: hole boundary defined by poly-lines, dashed line: Actual cut performed.

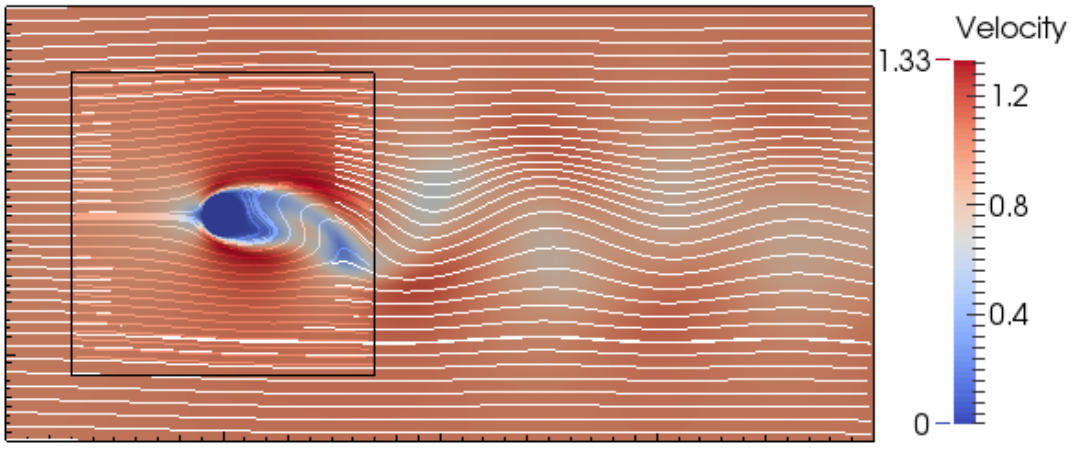
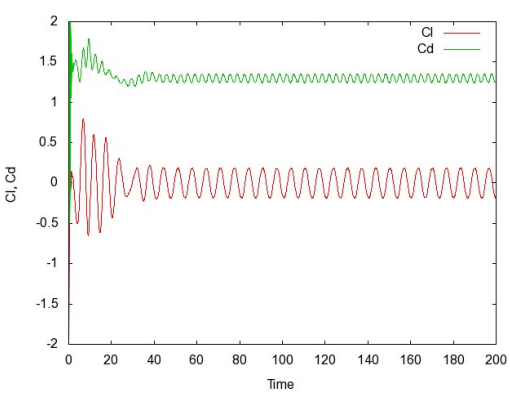
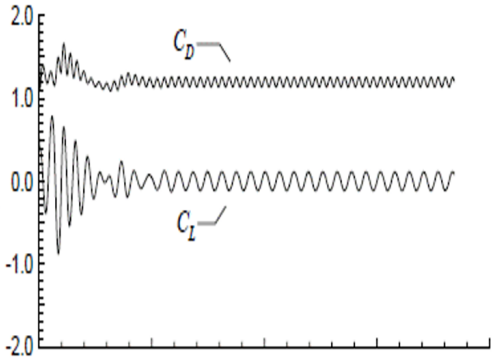


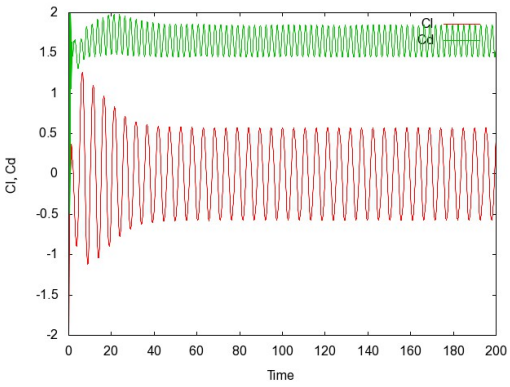
Figure 4.25: Velocity streamlines of the flow past an oscillating cylinder at t=300 using $f_e = 0.8$, with cut cell hole-cutting (Re=185).



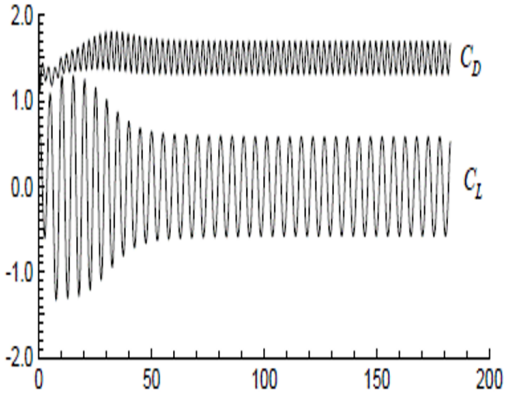
(a) Present, $f_e = 0.8$



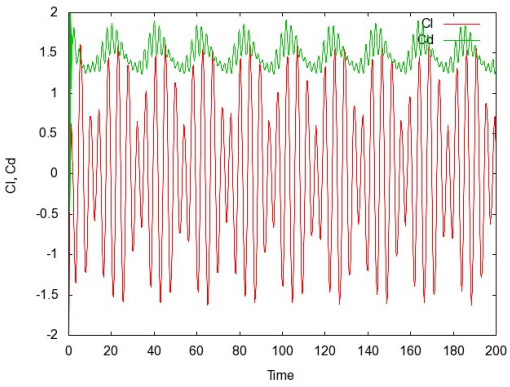
(b) Guilmineau & Queutey, $f_e = 0.8$



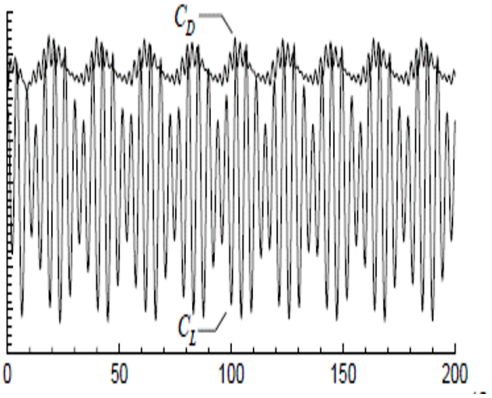
(c) Present, $f_e = 1.0$



(d) Guilmineau & Queutey, $f_e = 1.0$



(e) Present, $f_e = 1.2$



(f) Guilmineau & Queutey, $f_e = 1.2$

Figure 4.26: Plots of drag and lift coefficients over time for present results compared with published using cut cell hole-cutting (re=185).

4.2.4 Summary

Initially, the test was run using a fully explicit solver in artificial time for a Reynolds number of 185. This was found to be computationally slow due to the nature of explicit time marching. A new hybrid of artificial time integration methods was introduced, and results have shown that this has significantly increased the efficiency of the solver. Convergence in artificial time using implicit integration tends to be faster than when using explicit time integration. Since the bulk of the domain (covered by the background mesh) is now iterated implicitly, it is expected that convergence would be faster overall. The results showed that vortex shedding began earlier with the new hybrid. This difference in the results demonstrates the importance of the solvers accuracy for unsteady problems, which are dependent on time. For steady state flow problems, it will eventually converge to the solution, regardless of how many iterations this takes. But for unsteady problems, if the solver doesn't fully converge in artificial time, the solution at that time-step will not be accurate. This error will then be taken into the next time-step and will grow with every subsequent time-step. Given that convergence rates have now been increased, this difference in when vortex shedding begins could be indicating a convergence issue with the fully explicit method. But introducing an implicit scheme for a large proportion of the domain and interpolating solution data between this and the smaller explicit portion has resolved this issue. An alternative possibility is that the hybrid is introducing a small error which is inducing vortex shedding sooner. It was discovered that both overset solvers provided lower drag coefficient values than the original unstructured mesh solver from Chapter 3. However, in comparing the results to published numerical results, all three can be considered to agree well. Additionally, in a visual comparison with numerical published results for the drag and lift

over time, the implicit/explicit hybrid solver compares best in terms of the time vortex shedding begins. This leads to the conclusions that this overset solver is sufficiently accurate and efficient in simulating this type of flow problem. For this reason it was decided that the hybrid solver would undergo further development, leaving the slow fully explicit solver redundant. The hybrid solver was then tested for larger Reynolds numbers of 300, 500 and 1000. The results for the average drag coefficient and Strouhal number were in reasonable agreement with published numerical results.

Before further advances were made with the solver, an oscillating cylinder test was performed for a Reynolds number of 185. The oscillating case requires the use of the natural shedding frequency value, obtained from running the stationary cylinder case for the same Reynolds number. This means the overall accuracy of the results have some dependency on previous results being accurate. This is in addition to the importance of good convergence within artificial time, in order to produce an accurate time-dependent solution, as discussed previously. Despite these challenges, the solver performed well for the oscillating cylinder case; The drag and lift coefficient results were in good agreement with published results and it was observed that the solution data was smooth across the interface between the grids, meaning grid communication was good.

A novel hole-cutting method, employing the cut-cell method to perform the hole-cut was applied to the solver. Using this solver, both the stationary and oscillating cylinder cases were repeated for comparison. The solution obtained for the drag coefficient and Strouhal number has changed slightly with this hole-cut but matches the values from the solver using the simple hole-cutting method with fully explicit time-integration. This suggests that the presence of cut-cells is not negatively impacting on the accuracy of the solver for cases where re-cutting is not required. For the oscillating case, a noticeable error in the solution formed

due to the re-cutting of the hole boundary, which contained sharp edges. The issue was resolved by rounding the corners to the square hole boundary. Solutions obtained for the oscillating cases were found to transfer between grids well and were all in good agreement with published numerical results.

Chapter 5

2-Phase Flow Solver

5.1 2-Phase Flow Numerical Methods

5.1.1 Introduction

The next stage of the code development is to increase the capability of the solver to 2-phase (air & water) flow problems. This is important since it will enable the simulation of more complex flow features beyond the reach of the single fluid solver, using either the incompressible Navier-Stokes equations or Shallow Water equations. These complex features include a wave breaking and air entrapment. The purpose of this solver development is to be able to simulate real-world flow problems involving moving bodies such as a wave paddle device. Air entrapment is known to occur as a result of the movement of a wave paddle device, demonstrating the importance of the solvers capability to capture this. Within this chapter, only changes to the numerical methods for the unstructured 2-phase solver are outlined, for use as a single solver and for overlapping meshes in the overset solver. The background solver uses the in-house code Amazon-SC, which is a 2-phase solver. Therefore, the original code has been utilised here, with the

amendments for overset capability still in place. Chapters 3 and 4 should be referred to for full details on numerical methods implemented within the solvers, in addition to the changes discussed in this chapter.

5.1.2 Governing Equations

For 2-phase flow, the incompressible Navier-Stokes equations must be amended to include an additional density equation. In integral form, this new system of equations can be expressed as,

$$\frac{\partial}{\partial t}\iint_{\Omega} Q d\Omega + \oint\!\!\!\!\!\!\diagup_{s} F\dot{n}ds = \iint_{\Omega} B d\Omega \tag{5.1}$$

where Q is the vector of conserved variables, F is the flux vector and B is the vector for source terms for body forces. For inviscid, 2 dimensional problems, where the only body force is gravity,

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ p/\beta \end{bmatrix},$$

$$B = \begin{bmatrix} 0 \\ 0 \\ -\rho g \\ 0 \end{bmatrix},$$

$$F \cdot n = f^I n_x + g^I n_y, \tag{5.2}$$

$$f^I = \begin{bmatrix} \rho(u-u_b) \\ \rho u(u-u_b)+p \\ \rho v(u-u_b) \\ u \end{bmatrix}, \quad g^I = \begin{bmatrix} \rho(v-v_b) \\ \rho u(v-v_b) \\ \rho v(v-v_b)+p \\ v \end{bmatrix}$$

where ρ is the density, p is the pressure and u and v are the velocity components in the x and y directions, respectively. n_x and n_y are the x and y components of the normal vector, n of each cell side and g is the acceleration due to gravity. The primitive form of the conserved variables is given as $U = [\rho, u, v, p/\beta]^T$.

5.1.3 Spatial discretisation

For 2-phase flow, some changes to the spatial discretisation methods are required. The first of these lies within the flux function. Again Roe's flux approximation is adopted locally at each cell edge, as follows,

$$F^I_{i,j} = \frac{1}{2}[F^I(U^+_{i,j}) + F^I(U^-_{i,j}) - |A|(U^+_{i,j} - U^-_{i,j})] \tag{5.3}$$

$$|A| = R|\Lambda|L \tag{5.4}$$

where $U^+_{i,j}$ and $U^-_{i,j}$ are the reconstructed right and left states respectively of the cell edge between cells i and j, given in conservative form. The flux Jacobian [24] is now given as follows,

$$A = \frac{\partial(F \cdot n)}{\partial Q} = \begin{bmatrix} 0 & n_x & n_y & 0 \\ -u^2 n_x - uvn_y & 2un_x + vn_y & un_y & \beta n_x \\ -uvn_x - v^2 n_y & vn_x & un_x + 2vn_y & \beta n_y \\ -\frac{un_x}{\rho} - \frac{vn_y}{rho} & \frac{n_x}{\rho} & \frac{n_y}{\rho} & 0 \end{bmatrix} \tag{5.5}$$

which is evaluated by Roe's average state using primitive variables. The eigenvalues of the Jacobian are given in the matrix Λ, as follows,

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{bmatrix} \tag{5.6}$$

where $\lambda_{1,2} = un_x + vn_y$, $\lambda_{3,4} = \frac{1}{2}(un_x + vn_y \pm c)$, with $c = \sqrt{(un_x + vn_y)^2 + 4\beta\rho}$. The right (R) and left (L) eigenvectors, provided by [22] in general form are,

$$R = \begin{bmatrix} 1 & 0 & \lambda_3 & \lambda_4 \\ u & -n_y & u\lambda_3 + \frac{\beta n_x}{\rho} & u\lambda_4 + \frac{\beta n_x}{\rho} \\ v & n_x & v\lambda_3 + \frac{\beta n_y}{\rho} & v\lambda_4 + \frac{\beta n_y}{\rho} \\ 0 & 0 & -\frac{\lambda_4}{\rho} & -\frac{\lambda_3}{\rho} \end{bmatrix} \tag{5.7}$$

$$L = \begin{bmatrix} 1 + \frac{\rho\lambda_1}{\beta} & -\frac{\rho\lambda_1 n_x}{\beta} & -\frac{\rho\lambda_1 n_y}{\beta} & -\rho \\ un_y - vn_x & -n_y & n_x & 0 \\ -\frac{\rho\lambda_1\lambda_3}{\beta c} & \frac{\rho\lambda_3 n_x}{\beta c} & \frac{\rho\lambda_3 n_y}{\beta c} & \frac{\rho}{c} \\ \frac{\rho\lambda_1\lambda_4}{\beta c} & -\frac{\rho\lambda_4 n_x}{\beta c} & \frac{\rho\lambda_4 n_y}{\beta c} & \frac{\rho}{c} \end{bmatrix} \tag{5.8}$$

The variables within these eigenvectors and eigenvalues are the Roe-averaged values [93] as follows,

$$\rho = \sqrt{\rho^L \rho^R} \tag{5.9}$$

$$u = \frac{\sqrt{\rho^L}u^L + \sqrt{\rho^R}u^R}{\sqrt{\rho^L} + \sqrt{\rho^R}} \tag{5.10}$$

$$v = \frac{\sqrt{\rho^L}v^L + \sqrt{\rho^R}v^R}{\sqrt{\rho^L} + \sqrt{\rho^R}} \tag{5.11}$$

Boundary Conditions

Some changes also need to be made to the boundary condition implementation to include all four variables and gravity terms in the y-direction. For a solid boundary with reflection boundary conditions, the ghost cell variables can be obtained using,

$$\begin{aligned} \rho_g &= \rho_i \\ \mathbf{v}_g &= \mathbf{v}_i - 2(\mathbf{v}_i \cdot \mathbf{n})\mathbf{n} + 2(\mathbf{v}_b \cdot \mathbf{n})\mathbf{n} \\ p_g &= p_i - \rho_i g n_y |RO| \end{aligned} \tag{5.12}$$

where $\mathbf{v}_i$ is the vector containing both the x-directional velocity (u) and y-directional velocity (v), $\mathbf{v}_b$ is the body velocity vector and $|RO|$ is the distance between the centres of the ghost and interior cells. For the single unstructured solver, the body velocity term is excluded since it is not capable of simulating moving body problems. Similarly for an open boundary with transmissive boundary conditions, the ghost cell variables are obtained as follows,

$$\begin{aligned} \rho_g &= \rho_i \\ v_g &= v_i \\ p_g &= p_i - 0.5\rho_i g n_y |RO| \end{aligned} \tag{5.13}$$

Gradient Limiter

As previously discussed, for 2nd order spatial discretisation, a limiter should be applied to the cell gradients used for reconstruction at the cell edge. This is to prevent spurious oscillations in the solution. The previously implemented limiter is known to have some instability issues [94], along with many other limiters for unstructured grids [95]. While it appeared to be sufficient for the single fluid

solver, this is not the case for the 2-phase solver. An improved method, as proposed by Venkatakrishnan [96], is to replace equation 3.22 with,

$$\Phi_j(r_j) = \frac{r_j^2 + 2r_j + \epsilon^2}{r_j^2 + r_j + 2 + \epsilon^2} \tag{5.14}$$

to compute the limiter, Φ_j, through each cell edge with index j, where r_j is found through the following function, unchanged from the previous method,

$$r_j(U_j) = \begin{cases} (U_0^{max} - U_0)/(U_j - U_0) & if \quad U_j - U_0 > 0, \\ (U_0^{min} - U_0)/(U_j - U_0) & if \quad U_j - U_0 < 0, \\ 1 & if \quad U_j - U_0 = 0 \end{cases} \tag{5.15}$$

The constant, ϵ^2, is calculated using $\epsilon^2 = (Kh)^3$, where K is a user specified constant and h is the global average mesh size. The purpose of ϵ^2 is to prevent limiting smooth regions [97],[94]. In these regions, the constant will dominate the calculation of the limiter, reducing the value to 1 and leaving the gradient unchanged. If the constant is set to zero, the limiter will be active everywhere. Increasing the value from zero improves convergence, but if it is too large, the limiter will no longer be active and the solution may become unstable. The optimal value for the constant should be found through experimentation. Once a limiter has been computed for all 3 directions of a cell, again the minimum value is taken as the limiter for that cell.

Free Surface Capturing

For 2-phase flow problems, there is an interface between the fluids. This interface is treated as a free surface, acting under gravity. For this solver, the method chosen to deal with the free surface is the free surface capturing method [22],[23]. In this method, the interface is treated as contact discontinuity in the density

field [24], by giving each fluid region an appropriate density value. This allows the free surface to be captured automatically in the solution procedure, through the enforcement of conservation laws. In order to accurately capture the free surface, a sufficiently fine mesh is required. A high resolution Riemann solver should be used in conjunction with this approach, to ensure the interface remains sharp. In the current work, Roe's approximation [62] has been implemented.

5.1.4 Time Integration

For the unstructured solver, a first order explicit time-marching scheme is employed as follows,

$$\mathbf{U}_i^{n+1,m+1} = \mathbf{U}_i^{n+1,m} - \Delta\tau \frac{1}{A_i}(A_i \mathbf{I_0} \frac{\mathbf{U}_i^{n+1,m} - \mathbf{U}_i^n}{\Delta t} + \mathbf{R}_i^{*n+1,m}) \tag{5.16}$$

The original 4-stage Runge-Kutta time integration method has been discarded from the 2-phase solver due to instability issues, which were not present in the single fluid solver.

5.2 2-Phase Flow Validation Test Results

5.2.1 Introduction

In this section, results from benchmark tests are given. Firstly, the 2-phase capability of the single, unstructured solver was tested by performing a simple water column collapse simulation. Next, a small unstructured grid was placed over a Cartesian background grid and the test was repeated as an initial validation test for the overset solver.

5.2.2 Collapse of a Water Column

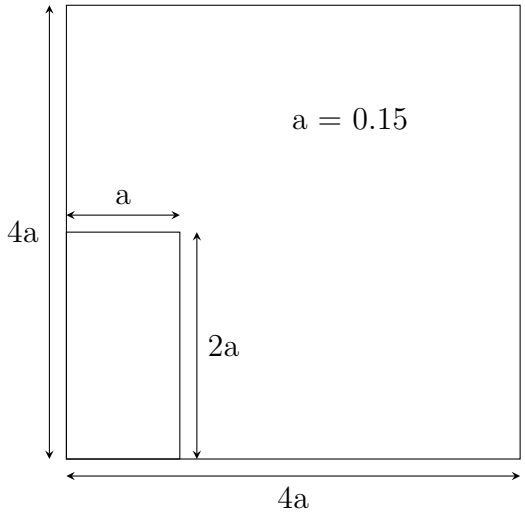


Figure 5.1: Initial setup of water column collapse problem.

The sudden collapse of a water column is a popular benchmark test for free surface flows due to its simplicity. This problem as illustrated by Figure 5.1, consists of a 0.6 × 0.6 tank covering the domain. A water column of dimensions 0.15 × 0.3 is positioned in the lower left corner of the tank. These dimensions were chosen to match those used for numerical results provided by [2]. Within this column the water density is set to 1000kg/m^3 and the density of air outside this region is 1kg/m^3. Initially the velocity is zero everywhere, and the pressure for a cell i, is calculated as,

$$p_i = 0.3\rho_i g/\beta + (0.3 - yc_i)\rho_i g/\beta \tag{5.17}$$

within the water column, where yc is the y-coordinate of the cell centre, and,

$$p_i = (0.6 - yc_i)\rho_i g/\beta \tag{5.18}$$

in the air region outside of the column. The only force acting within the tank is gravity. The boundary conditions used are a transmissive, open-top boundary, and reflective, solid walls for the 3 remaining sides of the tank.

Unstructured Solver

Initially the unstructured solver was tested using this problem. The mesh used is given by Figure 5.2. The geometry of the column has been considered in the mesh generation to ensure the free surface is represented correctly in the initial problem setup. The initial air (blue) and water (red) regions have been included in the illustration to demonstrate this.

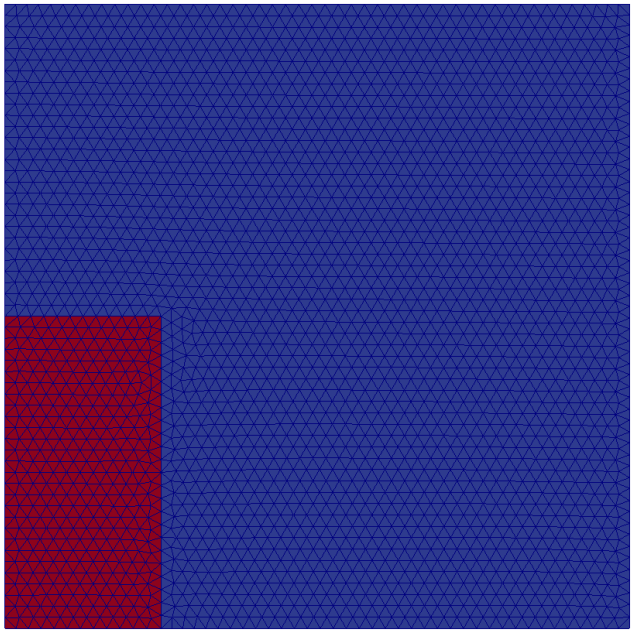


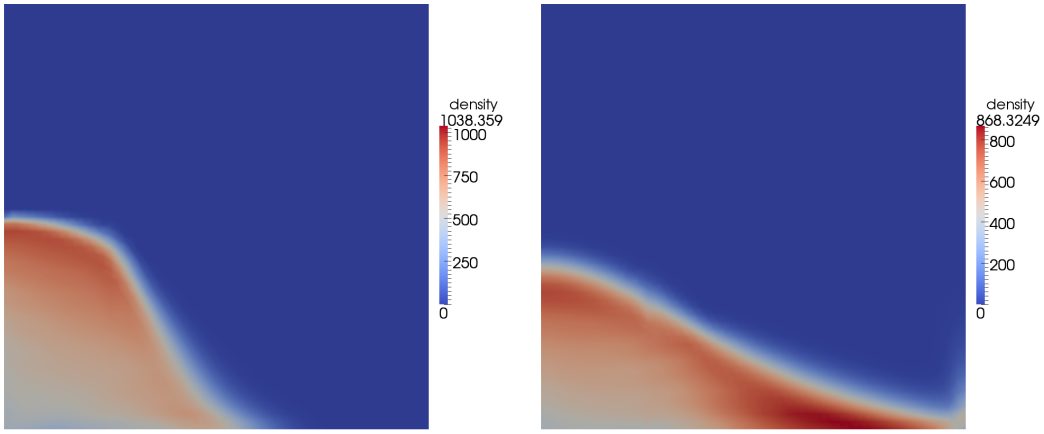
Figure 5.2: The unstructured mesh used in the collapse of a water column problem.

The mesh has a uniform distribution and a total of 5,275 cells. A real time-step of 0.00001 was used and the solver was given a run-time of 1s. A range of beta values from 1 to 10,000 were used to try to obtain reasonable convergence. In each case,

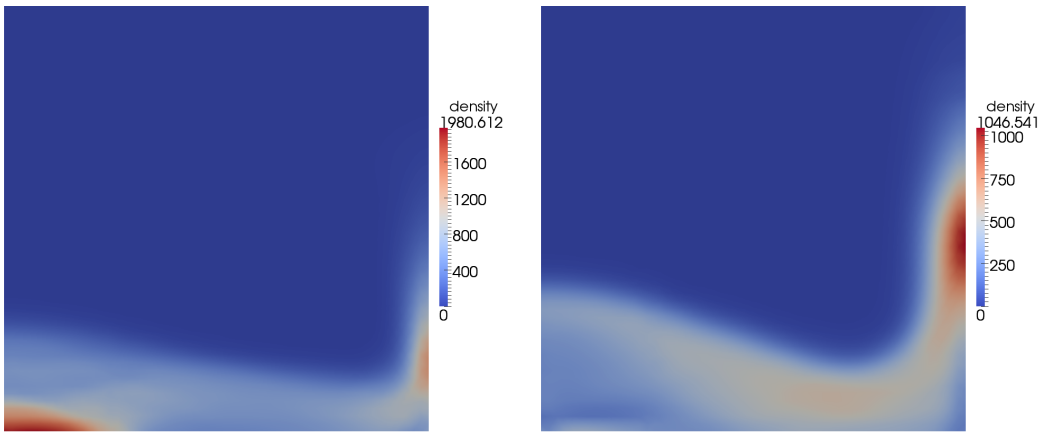
the solver failed to provide a stable solution. As discussed previously, with time-dependent problems such as this one, convergence in artificial time is fundamental in producing an accurate solution. Poor convergence in artificial time leads to an error building in the solution. This appears to be what is happening in this case. This Figure 5.3 shows the solution for density at various times throughout the simulation. The lack of convergence and resulting error formation is demonstrated by large densities that have formed in areas of the domain, which can be observed at t=0.3. Over time, the solution broke up and the free surface began to disperse. Over time, the solution broke up and the free surface began to disperse. This can be observed from t=0.3, where it is noticeable that the area under the free surface has started to increase. The solution shown at t=0.46, was the last output before the programme crashed. Here, the largest density is less than 448kg/m^3. In an attempt to improve convergence, a more stable limiter was applied, details of which are given in Section 5.1.3. This resulted in very little improvement. The poor convergence is thought to be caused by the use of the explicit artificial time-integration scheme. As well as this solution process working on a cell-by-cell basis, it also restricts the artificial time-step to maintain stability. As a result of these two factors, the convergence is much slower than that of implicit schemes.

Overset Solver

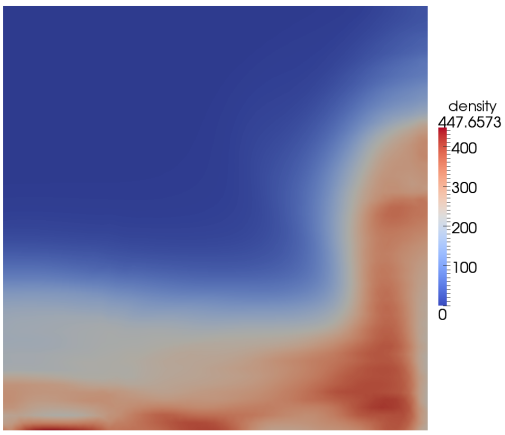
After failing to obtain a solution using the unstructured solver, the same problem was run using the overset solver. This was to observe whether a relatively small unstructured mesh would be able to provide a converged solution, with the aid of converged, interpolated data from the implicit background solver. For this test, the background grid covers the 0.6 $\times$ 0.6 region with 100 $\times$ 100 cells. A small overlapping mesh with dimensions 0.1 $\times$ 0.1 is positioned so that the free surface passes through it, as shown by Figure 5.4.



(a) t=0.1 (b) t=0.2

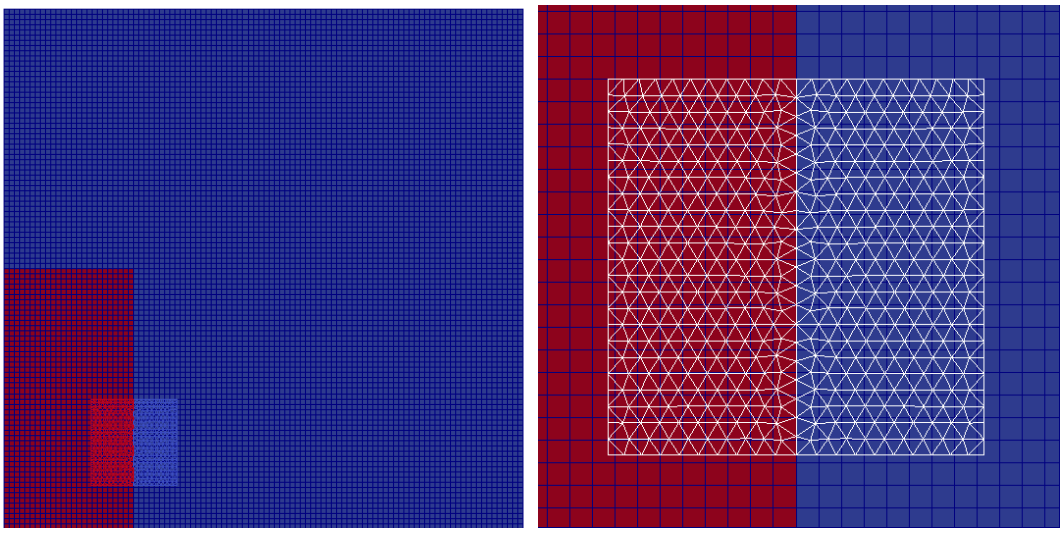


(c) t=0.3 (d) t=0.4



(e) t=0.46

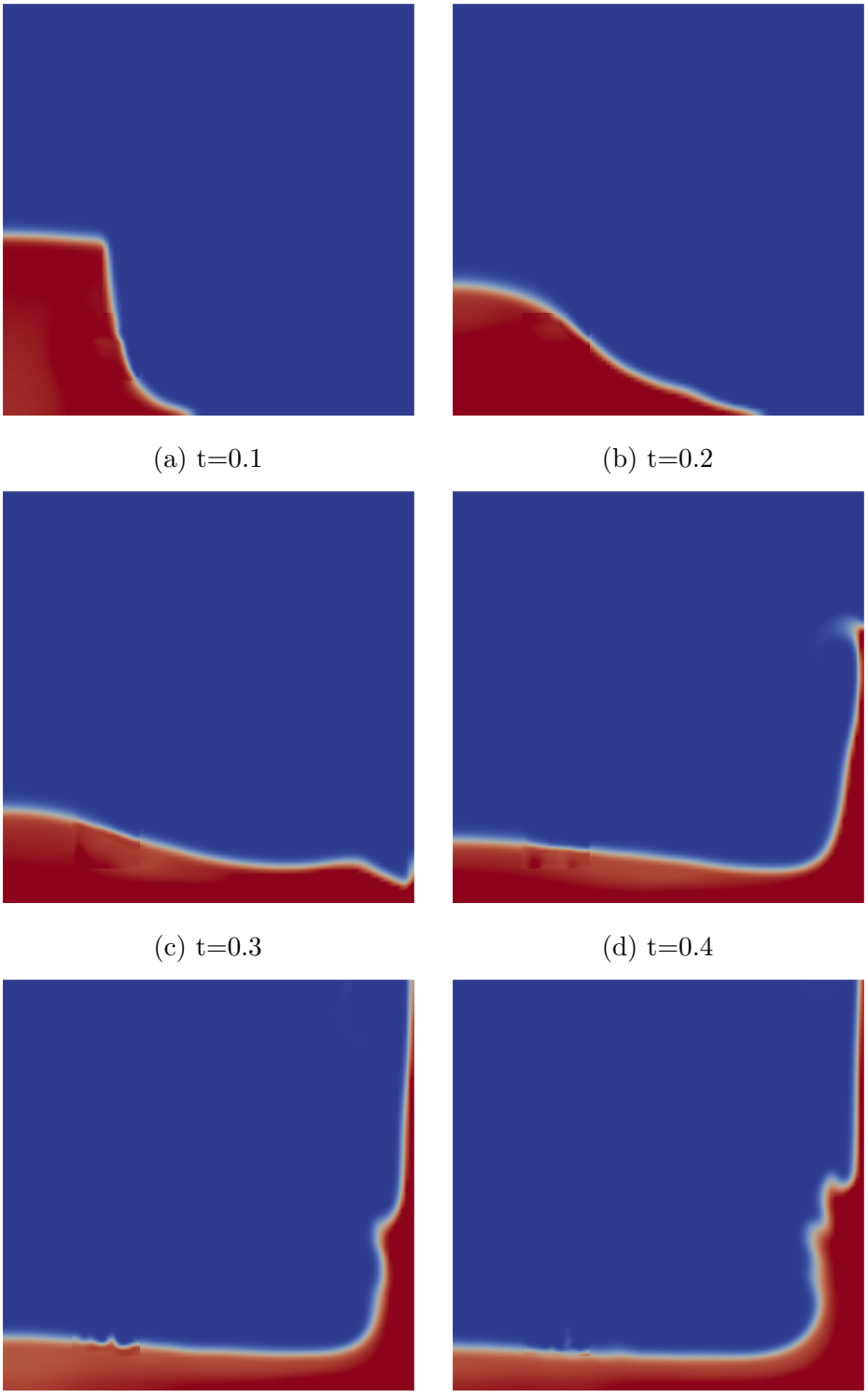
Figure 5.3: Collapse of water column simulation results at various times using the unstructured solver.



(a) Full Overlapping Meshes (b) Magnified View of Overlapping Mesh

Figure 5.4: The overlapping meshes used in the collapse of a water column problem.

Again, the initial air (blue) and water (red) regions have been included on the mesh. This is to demonstrate how the interface has been considered in the mesh generation. The smaller mesh is of uniform distribution with a total of 916 cells. The position of the overlapping mesh was chosen to observe the free surface passing throughout the simulation and ensure good communication is held between the meshes. A real time-step of 0.00001 was used, with an artificial time-step of 0.0001 used on the background mesh. A beta value of 1000 was found to provide the best convergence over the two meshes. The tolerance used for convergence was $10^{-}5$. Figures 5.5 and 5.6 show the solution at various times over the 1s runtime. In this case, the free surface is held throughout, providing a better solution than the unstructured solver. It can be observed that the free surface passes over the overlapping regions well, providing confidence that the overset method is functioning correctly. At t = 0.9, a region of air has been entrapped by water from an overturning wave, demonstrating the solver's ability to capture complex flow features. However, there does appear to be some unexplained disturbances

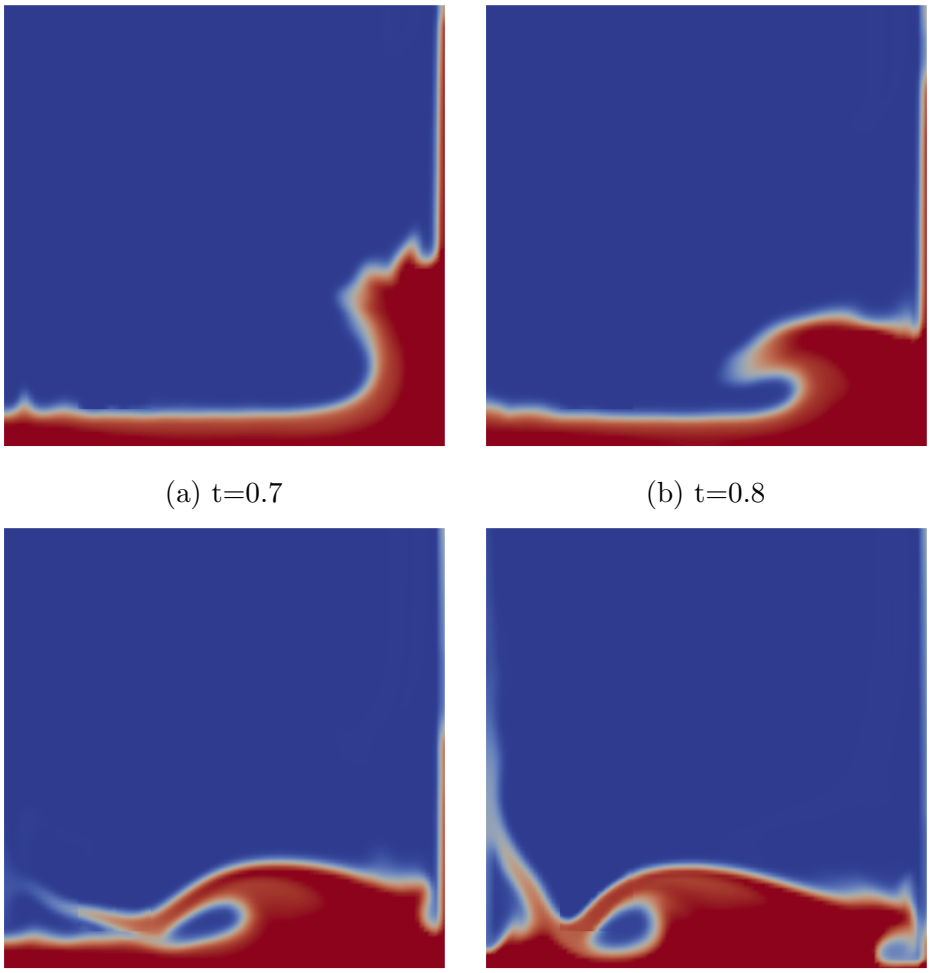


(a) t=0.1 (b) t=0.2

(c) t=0.3 (d) t=0.4

(e) t=0.5 (f) t=0.6

Figure 5.5: Collapse of water column simulation results at various times using the overset solver (Part 1).

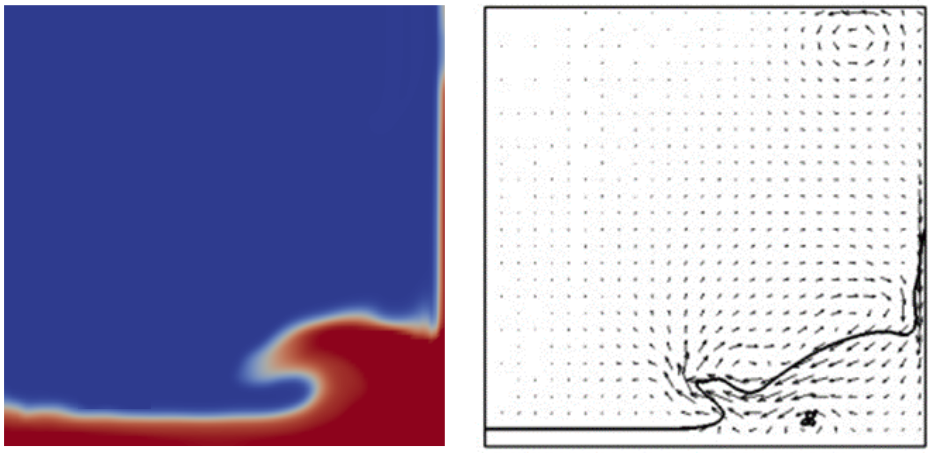


(a) t=0.7 (b) t=0.8

(c) t=0.9 (d) t=1.0

Figure 5.6: Collapse of water column simulation results at various times using the overset solver (Part 2).

of flow in the overlapping region, which can bee seen at t=0.5. Also, the result does not compare well to published results provided by [2]. Figure 5.7 shows a direct comparison of the current and published results, where the differences in the solution are clear. It appears that the poor convergence within the small overlapping region is having an impact on the whole solution.



(a) Present (b) Gu et al.

Figure 5.7: Comparison of solution to water column collapse at t=0.8 with published numerical results by Gu et al. [2].

5.2.3 Summary

A simple water column collapse problem has been used to test the unstructured and overset solvers. The unstructured solver was unable to obtain a converged solution in artificial time, which resulted in a break up of the solution. The overset solver did provide a solution over the whole runtime, but it did not compare well to published results. Problems with the unstructured overlapping solution were observed, leading to a conclusion that the convergence of the unstructured solver needs to be improved for 2-phase flow problems. To try and improve the convergence, a range of beta values were tested and an improved limiter was enforced. However, the problem remained. Since the explicit time-integration scheme used on the overlapping mesh is known to have convergence issues, a suitable solution would be to implement implicit time-integration on the overlapping mesh. Implicit time-integration was implemented on the background solver to reduce the overall CPU time required by the solver. Upon making this change, a vast improvement in the convergence rate was observed. An implicit solver should have the same effect on the unstructured, overlapping grid.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The research undergone in this thesis was aimed at developing an incompressible Navier-Stokes flow solver for moving body problems. A combination of structured and unstructured meshes were to be combined using an overset approach to allow for these body movements. The aim was achieved, with a contribution made to the research field, and was broken into three objectives.

Objective 1

The first objective was to implement a single fluid flow solver on an unstructured, 2D mesh using a cell centred finite volume approach. The purpose of this was to develop the solver that would later be used for the unstructured overlapping meshes in the overset approach. Unstructured meshes were chosen for their ability to handle complex geometries, making them appropriate for the overlapping meshes, where a solid body would need to be represented.

The objective was achieved by firstly writing a solver for the Shallow Water equations and then developing it into an incompressible Navier-Stokes solver.

This was to reduce the amount of new technical knowledge that needed to be applied to the solver in the immediate stages. Benchmark validation tests were performed for both systems of equations and in each case, the results were found to be in good agreement with published numerical results.

Objective 2

The second objective was to develop the overset meshing capability of the flow solver through the use of algorithms for automatic detection of overlapping regions between block meshes and data interpolation for boundary conditions. Again, a systematic approach was taken to achieve this objective. First, the unstructured mesh solver (from objective 1) was combined with the in-house Cartesian grid solver, AMAZON-SC. Underneath the overlapping grid, a hole must be cut from the background grid, leaving a small overlap. A simple but unsophisticated approach was taken for the hole-cutting process initially. It had many limitations and was used as a building block towards a more sophisticated approach, similarly to the use of the Shallow Water equations for objective1.

Next, the simple hole-cutting process was replaced by a novel approach; whereby the hole-cut would be performed using the Cartesian cut-cell method. This is an existing cutting method, but for a different purpose, which is to cut solid objects from a Cartesian grid. It has not been used for hole-cutting previously within the literature. Some modifications were made to the treatment of cut-cells to allow for a hole-cut rather than solid object representation. The purpose of this new approach was to establish an automatic hole-cutting method, which would provide a simple and efficient alternative to existing hole-cutting techniques. It differs from existing techniques in the way that the cut is performed; it slices through cells rather than around them. The method uses a set

of poly-lines to define where the cut will be made, meaning there is no need for any search algorithms to create the hole. These poly-lines are automatically updated with the overlapping grid position for moving body problems.

Again, benchmark validation tests were performed at each development stage; using the simple and new hole-cutting methods. The solver was able to perform moving body simulations and results compared well to published numerical results.

A problem that was encountered whilst working towards this objective was the efficiency of the solver. The CPU time required to run a simulation of the flow past a stationary cylinder case was significantly large. The solution proposed, was to change the time integration scheme for the Cartesian background solver from explicit to implicit. This created a new hybrid of integration schemes over the overset grids. Due to the nature of the convergence of each scheme, adjustments to the artificial time integration process needed to be made. The same validation test was then repeated and with this new hybrid of integration schemes, the CPU time required was reduced by a factor of approximately 9.

Objective 3

The third objective was to develop the solver for two-phase (air & water) flow simulations. The purpose of this was to allow for more complex flow simulations, which more accurately describe real-world flow problems. Unfortunately, this objective has not been completed. The relevant changes have been made to both the single, unstructured solver and the overset solver to allow for 2-phase flow. However, upon testing the solvers, a problem was encountered. The solution appears to be unable to converge in artificial time, which results in fluctuating density values over the domain and eventually a loss of the free surface. A new

gradient limiter was implemented to improve the convergence, but this did not result in a significant improvement. It is thought that the convergence issues are a result of the explicit time integration scheme used on the unstructured meshes. A proposed solution is to implement an implicit time integration scheme on the overlapping meshes. This was disregarded in the earlier stages for the solver development. It was thought that an explicit time-integration scheme would be more advantageous to the solver in the future, since it is far less difficult to parallelise than an implicit scheme, especially given the unstructured form of the data. Implementing an implicit scheme would also require the use of complicated sparse matrices, which could be avoided with an explicit scheme. It was thought that the much simpler explicit method would be sufficient for small overlapping regions, where any effects on solution convergence should be minimised. Although this was the case for the single fluid solver, an implicit solver is now considered more suitable for the 2-phase flow solver. Unfortunately, time restrictions have not allowed for this suggested solution to be implemented and tested.

Summary

Overall, the research project went well, with a good contribution to the research field being made. The application of the Cartesian cut-cell approach to hole-cutting proved successful for the single fluid flow solver. The location of the hole-cut is updated automatically, along with the locations of cells that maintain the communication between the meshes. This eliminates the need for user input throughout the solution process, as required by existing explicit techniques. It is also a very simple cutting process, which is already well established and understood within the field. This gives it appeal over the complex, implicit techniques currently in use. Unfortunately, the method could not be fully tested within the

2-phase solver, due to convergence issues in the initial validation tests. However, there should be no reason for the extension to 2-phase flow to affect the performance of the hole-cutting technique.

The numerical solver was developed for use within the CMMFA to aid their current and future research. In its current form, the overset solver is capable of capturing many flow problems of interest to the CMMFA and the research field in general. There are may potential industrial applications for this solver, including aeronautical/aerospace engineering, the automotive industry and environmental science and engineering. The CMMFA are currently interested in developing a hybrid numerical wave tank (NWT) facility, where a single fluid, free surface solver can be applied to a large region in a NWT for wave generation and propagation, while the more computationally intensive multi-phase flow solver is only needed around the region with breaking waves and floating structures.

6.2 Future Work

The future work to be carried out includes some further code development and simulations, as follows:

- Allow for rotational movement in the single fluid solver. The single fluid solver has produced good results for both stationary and moving body problems. However, rotational movement has yet to be applied. Currently the body velocity is applied as a constant over the overlapping mesh. To allow for rotational movement, this would need to be changed to a local velocity, stored at each cell centre. This could then be tested using a flow past a rotating aerofoil problem.

- Implement an implicit time integration scheme on the unstructured solver,

for 2-phase flow problems. The 2-phase flow solver was unable to obtain a converged solution in artificial time using an explicit approach. It is thought that an implicit method will solve this issue. The collapse of a water column test should then be repeated to observe any improvements in the convergence.

- Validate the 2-phase solver for moving body problems. This is only possible once the convergence issues have been resolved. A suitable test is a water entry of a wedge problem.

Upon completion of the above, the solver would be capable of simulating more complex real-world problems, relevant to the the University's current research within the CMMFA as well as further industrial applications. Examples include, the simulation of a wave paddle device for generating renewable energy, such as the Aquamarine Power Oyster Paddle Device [98].

References

[1] E. Guilmineau and P. Queutey, "A numerical simulation of vortex shedding from an oscillating circular cylinder," *Journal of Fluids and Structures*, vol. 16, no. 6, pp. 773–794, 2002.

[2] H. Gu, D. Causon, C. Mingham, and L. Qian, "Development of a free surface flow solver for the simulation of wave/body interactions," *European Journal of Mechanics-B/Fluids*, vol. 38, pp. 1–17, 2013.

[3] Z. Z. Hu, D. M. Causon, C. G. Mingham, and L. Qian, "A cartesian cut cell free surface capturing method for 3d water impact problems," *International Journal for Numerical Methods in Fluids*, vol. 71, no. 10, pp. 1238–1259, 2013.

[4] Z. Z. Hu, D. M. Causon, C. G. Mingham, and L. Qian, "Numerical simulation of floating bodies in extreme free surface waves," *Nat. Hazards Earth Syst. Sci.*, vol. 11, no. 2, pp. 519–527, 2011.

[5] R. Eymard, T. Gallout, and R. Herbin, *Finite volume methods*, vol. 7, pp. 713–1018. Elsevier, 2000.

[6] D. Mavriplis, "Unstructured grid techniques," *Annual Review of Fluid Mechanics*, vol. 29, no. 1, pp. 473–514, 1997.

[7] J. Cai, H. M. Tsai, and F. Liu, "A parallel viscous flow solver on multi-block overset grids," *Computers & fluids*, vol. 35, no. 10, pp. 1290–1301, 2006.

[8] D. Chandar, J. Sitaraman, and D. Mavriplis, "Gpu parallelization of an unstructured overset grid incompressible navier-stokes solver for moving bodies," in *50 th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, Nashville, TN*.

[9] R. Koomullil, G. Cheng, B. Soni, R. Noack, and N. Prewitt, "Moving-body simulations using overset framework with rigid body dynamics," *Mathematics and Computers in Simulation*, vol. 78, no. 5, pp. 618–626, 2008.

[10] H. Tang, S. Casey Jones, and F. Sotiropoulos, "An overset-grid method for 3d unsteady incompressible flows," *Journal of Computational Physics*, vol. 191, no. 2, pp. 567–600, 2003.

[11] P. Lin, "A fixed-grid model for simulation of a moving body in free surface flows," *Computers & fluids*, vol. 36, no. 3, pp. 549–561, 2007.

[12] W. M. Chan, "Overset grid technology development at nasa ames research center," *Computers & Fluids*, vol. 38, no. 3, pp. 496–503, 2009.

[13] P. McDonald, "The computation of transonic flow through two-dimensional gas turbine cascades," in *ASME 1971 International Gas Turbine Conference and Products Show*, pp. V001T01A089–V001T01A089, American Society of Mechanical Engineers.

[14] J. H. Ferziger and M. Peri, *Computational methods for fluid dynamics*. 1997.

[15] A. J. Chorin, "Numerical solution of the navier-stokes equations," *Mathematics of computation*, vol. 22, no. 104, pp. 745–762, 1968.

[16] D. L. Brown, R. Cortez, and M. L. Minion, "Accurate projection methods for the incompressible navierstokes equations," *Journal of Computational Physics*, vol. 168, no. 2, pp. 464–499, 2001.

[17] A. J. Chorin, "A numerical method for solving incompressible viscous flow problems," *Journal of computational physics*, vol. 2, no. 1, pp. 12–26, 1967.

[18] C. Chan and K. Anastasiou, "Solution of incompressible flows with or without a free surface using the finite volume method on unstructured triangular meshes," *International journal for numerical methods in fluids*, vol. 29, no. 1, pp. 35–57, 1999.

[19] J. Glimm, J. Grove, B. Lindquist, O. A. McBryan, and G. Tryggvason, "The bifurcation of tracked scalar waves," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 1, pp. 61–79, 1988.

[20] C. Lemos, "A simple numerical technique for turbulent flows with free surfaces," *International journal for numerical methods in fluids*, vol. 15, no. 2, pp. 127–146, 1992.

[21] J. Farmer, L. Martinelli, and A. Jameson, "Fast multigrid method for solving incompressible hydrodynamic problems with free surfaces," *AIAA journal*, vol. 32, no. 6, pp. 1175–1182, 1994.

[22] F. J. Kelecy and R. H. Pletcher, "The development of a free surface capturing approach for multidimensional free surface flows in closed containers," *Journal of Computational Physics*, vol. 138, no. 2, pp. 939–980, 1997.

[23] D. Pan and C.-H. Chang, "The capturing of free surfaces in incompressible multi-fluid flows," *International Journal for Numerical Methods in Fluids*, vol. 33, no. 2, pp. 203–222, 2000.

[24] L. Qian, D. Causon, C. Mingham, and D. Ingram, "A free-surface capturing method for two fluid flows with moving bodies," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, vol. 462, no. 2065, pp. 21–42, 2006.

[25] C. W. Hirt and B. D. Nichols, "Volume of fluid (vof) method for the dynamics of free boundaries," *Journal of computational physics*, vol. 39, no. 1, pp. 201–225, 1981.

[26] Y. Zhao, H. Hui Tan, and B. Zhang, "A high-resolution characteristics-based implicit dual time-stepping vof method for free surface flow simulation on unstructured grids," *Journal of Computational Physics*, vol. 183, no. 1, pp. 233–273, 2002.

[27] D. J. Mavriplis, "Mesh generation and adaptivity for complex geometries and flows," *Handbook of Computational Fluid Mechanics*, pp. 417–459, 1996.

[28] R. Mittal and G. Iaccarino, "Immersed boundary methods," *Annu. Rev. Fluid Mech.*, vol. 37, pp. 239–261, 2005.

[29] C. S. Peskin, "Flow patterns around heart valves: A numerical method," *Journal of Computational Physics*, vol. 10, no. 2, pp. 252–271, 1972.

[30] B. E. Griffith, R. D. Hornung, D. M. McQueen, and C. S. Peskin, "An adaptive, formally second order accurate version of the immersed boundary method," *Journal of Computational Physics*, vol. 223, no. 1, pp. 10–49, 2007.

[31] T. Ye, R. Mittal, H. Udaykumar, and W. Shyy, "An accurate cartesian grid method for viscous incompressible flows with complex immersed boundaries," *Journal of computational physics*, vol. 156, no. 2, pp. 209–240, 1999.

[32] I. Borazjani, L. Ge, T. Le, and F. Sotiropoulos, "A parallel overset-curvilinear-immersed boundary framework for simulating complex 3d incompressible flows," *Computers & fluids*, vol. 77, pp. 76–96, 2013.

[33] J. Lee, J. Kim, H. Choi, and K.-S. Yang, "Sources of spurious force oscillations from an immersed boundary method for moving-body problems," *Journal of computational physics*, vol. 230, no. 7, pp. 2677–2695, 2011.

[34] M. Aftosmis, J. Melton, and M. Berger, "Adaptation and surface modeling for cartesian mesh methods,"

[35] D. DeZeeuw and K. G. Powell, "An adaptively refined cartesian mesh solver for the euler equations," *Journal of Computational Physics*, vol. 104, no. 1, pp. 56–68, 1993.

[36] G. Yang, D. Causon, D. Ingram, R. Saunders, and P. Batten, "A cartesian cut cell method for compressible flows part a: Static body problems," *Aeronautical Journal*, vol. 101, no. 1002, pp. 47–56, 1997.

[37] G. Yang, D. Causon, D. Ingram, R. Saunders, and P. Batten, "A cartesian cut cell method for compressible flows part b: moving body problems," *Aeronautical Journal*, vol. 101, no. 1002, pp. 57–65, 1997.

[38] D. M. Causon, D. M. Ingram, C. G. Mingham, G. Yang, and R. V. Pearson, "Calculation of shallow water flows using a cartesian cut cell approach," *Advances in water resources*, vol. 23, no. 5, pp. 545–562, 2000.

[39] D. M. Ingram, D. M. Causon, and C. G. Mingham, "Developments in cartesian cut cell methods," *Mathematics and Computers in Simulation*, vol. 61, no. 3, pp. 561–572, 2003.

[40] S. R. Richardson, *Numerical simulation of impulsive wave overtopping events resulting from landslides*. Thesis, 2002.

[41] D. K. Clarke, H. Hassan, and M. Salas, "Euler calculations for multielement airfoils using cartesian grids," *AIAA journal*, vol. 24, no. 3, pp. 353–358, 1986.

[42] F. Gao, D. M. Ingram, D. M. Causon, and C. G. Mingham, "The development of a cartesian cut cell method for incompressible viscous flows," *International Journal for Numerical Methods in Fluids*, vol. 54, no. 9, pp. 1033–1053, 2007.

[43] D. J. Mavriplis, "Unstructured-mesh discretizations and solvers for computational aerodynamics," *AIAA journal*, vol. 46, no. 6, pp. 1281–1298, 2008.

[44] D. J. Mavriplis, "An advancing front delaunay triangulation algorithm designed for robustness," *Journal of Computational Physics*, vol. 117, no. 1, pp. 90–101, 1995.

[45] R. Lhner and P. Parikh, "Generation of threedimensional unstructured grids by the advancingfront method," *International Journal for Numerical Methods in Fluids*, vol. 8, no. 10, pp. 1135–1149, 1988.

[46] D. J. Mavriplis, "Adaptive mesh generation for viscous flows using triangulation," *Journal of computational Physics*, vol. 90, no. 2, pp. 271–291, 1990.

[47] S. Rebay, "Efficient unstructured mesh generation by means of delaunay triangulation and bowyer-watson algorithm," *Journal of Computational Physics*, vol. 106, no. 1, pp. 125–138, 1993.

[48] H. Jasak and Z. Tukovic, "Automatic mesh motion for the unstructured finite volume method," *Transactions of FAMENA*, vol. 30, no. 2, pp. 1–20, 2006.

[49] K. Matsushima, M. Murayama, and K. Nakahashi, "Unstructured dynamic mesh for large movement and deformation," in *40th AIAA Aerospace Sciences Meeting & Exhibit*, p. 122.

[50] R. Panahi and M. Shafieefar, "Towards a numerical hydrodynamics laboratory by developing an overlapping mesh solver based on a moving mesh solver; verification and application," *Applied Ocean Research*, vol. 32, no. 3, pp. 308–320, 2010.

[51] T. Craft, H. Iacovides, and A. Skillen, "A new overset grid algorithm applied to the simulation of flows involving complex geometries,"

[52] R. L. Meakin, "Object x-rays for cutting holes in composite overset structured grids," *AIAA paper*, vol. 2537, p. 2001, 2001.

[53] N. Kim and W. M. Chan, "Automation of hole-cutting for overset grids using the x-rays approach," in *20th AIAA Computational Fluid Dynamics Conference, AIAA*, vol. 3052.

[54] W. M. Chan, N. Kim, and S. A. Pandya, "Advances in domain connectivity for overset grids using the x-rays approach," 2012.

[55] Y. Lee, *On overset grids connectivity and vortex tracking in rotorcraft cfd.* University of Maryland, College Park, 2008.

[56] Y. Lee and J. D. Baeder, "Implicit hole cuttinga new approach to overset grid connectivity," *AIAA paper*, vol. 4128, p. 2003, 2003.

[57] P.-O. Persson, "Distmesh - a simple mesh generator in matlab." `http://persson.berkeley.edu/distmesh/`. Accessed: 23-02-2017.

[58] P.-O. Persson and G. Strang, "A simple mesh generator in matlab," *SIAM review*, vol. 46, no. 2, pp. 329–345, 2004.

[59] S. O. R. Fedkiw, "Level set methods and dynamic implicit surfaces," 2003.

[60] P. Garcia-Navarro, M. Hubbard, and A. Priestley, "Genuinely multidimensional upwinding for the 2d shallow water equations," *Journal of Computational Physics*, vol. 121, no. 1, pp. 79–93, 1995.

[61] P. B. Garca and P. G. Navarro, "Upwind numerical techiniques applied to the simulation of dam break flows," in *VII Jornadas Zaragoza-Pau de Matemtica Aplicada y estadstica: Jaca (Huesca), 17-18 de septiembre de 2001*, pp. 145–152, Prensas Universitarias de Zaragoza.

[62] P. L. Roe, "Approximate riemann solvers, parameter vectors, and difference schemes," *Journal of computational physics*, vol. 43, no. 2, pp. 357–372, 1981.

[63] K. Anastasiou and C. Chan, "Solution of the 2d shallow water equations using the finite volume method on unstructured triangular meshes," *International Journal for Numerical Methods in Fluids*, vol. 24, no. 11, pp. 1225–1245, 1997.

[64] P. Brufau and P. Garcia-Navarro, "Two-dimensional dam break flow simulation," *International Journal for Numerical Methods in Fluids*, vol. 33, no. 1, pp. 35–57, 2000.

[65] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM national conference*, pp. 517–524, ACM.

[66] T. Barth and D. Jespersen, "The design and application of upwind schemes on unstructured meshes," in *27th Aerospace sciences meeting*, p. 366.

[67] P. Roe, "Characteristic-based schemes for the euler equations," *Annual review of fluid mechanics*, vol. 18, no. 1, pp. 337–365, 1986.

[68] A. L. Gaitonde, "A dual-time method for two-dimensional unsteady incompressible flow calculations," *International Journal for Numerical Methods in Engineering*, vol. 41, no. 6, pp. 1153–1166, 1998.

[69] H. Gough, A. L. Gaitonde, and D. P. Jones, "A dual-time central-difference interface-capturing finite volume scheme applied to cavitation modelling," *International Journal for Numerical Methods in Fluids*, vol. 66, no. 4, pp. 452–485, 2011.

[70] A. Jameson, W. Schmidt, and E. Turkel, "Numerical solutions of the euler equations by finite volume methods using runge-kutta time-stepping schemes," *AIAA paper*, vol. 1259, p. 1981, 1981.

[71] F. Alcrudo and P. Garcia-Navarro, "A high-resolution godunov-type scheme in finite volumes for the 2d shallow-water equations," *International Journal for Numerical Methods in Fluids*, vol. 16, no. 6, pp. 489–505, 1993.

[72] C. Mingham and D. Causon, "High-resolution finite-volume method for shallow water flows," *Journal of Hydraulic Engineering*, vol. 124, no. 6, pp. 605–614, 1998.

[73] J.-S. Lai, G.-F. Lin, and W.-D. Guo, "An upstream flux-splitting finite-volume scheme for 2d shallow water equations," *International journal for numerical methods in fluids*, vol. 48, no. 10, pp. 1149–1174, 2005.

[74] C. Zoppou and S. Roberts, "Numerical solution of the two-dimensional unsteady dam break," *Applied Mathematical Modelling*, vol. 24, no. 7, pp. 457–475, 2000.

[75] U. Ghia, K. N. Ghia, and C. Shin, "High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method," *Journal of computational physics*, vol. 48, no. 3, pp. 387–411, 1982.

[76] Z. Ma, L. Qian, D. Causon, H. Gu, and C. Mingham, "A cartesian ghostcell multigrid poisson solver for incompressible flows," *International journal for numerical methods in engineering*, vol. 85, no. 2, pp. 230–246, 2011.

[77] L. Qian, D. Causon, D. Ingram, C. Mingham, and J. Zhou, "A cartesian cut cell method for incompressible viscous flows," in *Proc. ECCOMAS CFD*, pp. 4–7.

[78] A. Ongoren and D. Rockwell, "Flow structure from an oscillating cylinder part 1. mechanisms of phase shift and recovery in the near wake," *Journal of fluid Mechanics*, vol. 191, pp. 197–223, 1988.

[79] A.-H. Pham, C.-Y. Lee, J.-H. Seo, H.-H. Chun, H.-J. Kim, H.-S. Yoon, J.-H. Kim, D.-W. Park, and I.-R. Park, "Laminar flow past an oscillating circular cylinder in cross flow," *Journal of Marine Science and Technology*, vol. 18, no. 3, pp. 361–368, 2010.

[80] M.-H. Chung, "Cartesian cut cell approach for simulating incompressible flows with rigid bodies of arbitrary shape," *Computers & Fluids*, vol. 35, no. 6, pp. 607–623, 2006.

[81] B. Fornberg, "A numerical study of steady viscous flow past a circular cylinder," *Journal of Fluid Mechanics*, vol. 98, no. 04, pp. 819–855, 1980.

[82] D. Hartmann, M. Meinke, and W. Schrder, "An adaptive multilevel multigrid formulation for cartesian hierarchical grid methods," *Computers & Fluids*, vol. 37, no. 9, pp. 1103–1125, 2008.

[83] X.-Y. Lu and C. Dalton, "Calculation of the timing of vortex formation from an oscillating cylinder," *Journal of Fluids and Structures*, vol. 10, no. 5, pp. 527–541, 1996.

[84] Z.-L. Kang, C. Yan, J. Yu, and Y.-Y. Fang, "A fast and reliable overset unstructured grids approach," *Acta Mechanica Sinica*, vol. 29, no. 2, pp. 149–157, 2013.

[85] P. Ridley, "Guide to partitioning unstructured meshes for parallel computing," 2010.

[86] T. M. Burton and J. K. Eaton, "Analysis of a fractional-step method on overset grids," *Journal of Computational Physics*, vol. 177, no. 2, pp. 336–364, 2002.

[87] H. Lomax and D. Pan, "A new approximate lu factorization scheme for the reynolds-averaged navier-stokes equations," *AIAA Journal*, vol. 26, no. 2, pp. 163–171, 1988.

[88] L. Li, S. Sherwin, and P. W. Bearman, "A moving frame of reference algorithm for fluid/structure interaction of rotating and translating bodies," *International Journal for Numerical Methods in Fluids*, vol. 38, no. 2, pp. 187–206, 2002.

[89] W. Bai, C. G. Mingham, D. M. Causon, and L. Qian, "Finite volume simulation of viscous free surface waves using the cartesian cut cell approach,"

International Journal for Numerical Methods in Fluids, vol. 63, no. 1, pp. 69–95, 2010.

[90] B. N. Rajani, A. Kandasamy, and S. Majumdar, "Numerical simulation of laminar flow past a circular cylinder," *Applied Mathematical Modelling*, vol. 33, no. 3, pp. 1228–1247, 2009.

[91] S. Mittal and R. Balachandar, "On the inclusion of thre dimensional effects in simulations of two-dimensional bluff body wake flows," 1997.

[92] L. Qian and M. Vezza, "A vorticity-based method for incompressible unsteady viscous flows," *Journal of computational physics*, vol. 172, no. 2, pp. 515–542, 2001.

[93] A. Yang, S. Chen, L. Yang, and X. Yang, "An upwind finite volume method for incompressible inviscid free surface flows," *Computers & Fluids*, vol. 101, pp. 170–182, 2014.

[94] M. S. Darwish and F. Moukalled, "Tvd schemes for unstructured grids," *International Journal of Heat and Mass Transfer*, vol. 46, no. 4, pp. 599–611, 2003.

[95] M. Berger, M. J. Aftosmis, and S. M. Murman, "Analysis of slope limiters on irregular grids," *AIAA paper*, vol. 490, no. 2005, pp. 1–22, 2005.

[96] V. Venkatakrishnan, "On the accuracy of limiters and convergence to steady state solutions," in *31st Aerospace Sciences Meeting*, p. 880.

[97] K. Michalak and C. Ollivier-Gooch, "Limiters for unstructured higher-order accurate solutions of the euler equations," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, p. 776.

[98] "Aquamarine power ltd." `http://www.aquamarinepower.com/`. Accessed: 20-08-2016.